Modelo de enseñanza basada en casos: de los tutores inteligentes a los videojuegos

TESIS DOCTORAL

Pedro Pablo Gómez Martín

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Octubre 2.007

Modelo de enseñanza basada en casos: de los tutores inteligentes a los videojuegos

Memoria que presenta para optar al título de Doctor en Informática

Pedro Pablo Gómez Martín

Dirigida por el doctor

Pedro Antonio González Calero

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Octubre 2.007

A todo aquel que
confió
en que lo conseguiría

Agradecimientos

De gente bien nacida es agradecer los beneficios que reciben, y uno de los pecados que más a Dios ofende es la ingratitud.

Don Quijote de la Mancha, Miguel de Cervantes

¡Válame Dios, y con cuánta gana debes de estar esperando ahora, lector ilustre, o quier plebeyo, este prólogo, creyendo hallar en él agasajos, halagos y lisonjas para aquellos que en algún punto se acercaron al autor de este documento! Pues en verdad que te he de dar este contento; que puesto que el amparo y la merced despiertan la gratitud en los más humildes pechos, el mío no ha de padecer excepción a esta regla.

Como sabes, llegar aquí ha sido largo y difícil. Y, al hacer camino al andar, muchos sois los que os habéis cruzado a lo largo de mi peregrinaje, moldeando mi forma de andar, sujetándome al tropezar y animándome al dudar.

De todos vosotros, has sido tú, Marco Antonio, quien ha viajado todo el tiempo a mi lado. De la mano hemos bogado por aguas inexploradas para, al fin, llegar hoy a buen puerto. ¿Cómo no agradecerte que remaras a mi lado para, entre los dos, conseguir alcanzar nuestra meta? ¿Cómo no alabar tu apoyo incondicional, las conversaciones clarificadoras y los ajustes al timón que han conseguido este final? Me dejas sin palabras. Y, si las tuviera, no podrían decirte nada que tú no supieras ya.

Pedro, tú siempre has estado vigilándonos de cerca, para que no nos desviáramos y siguiéramos nuestra natural tendencia a dirigir los pasos por otros senderos. A pesar de todo, la tentación creada por algunos de ellos fue demasiado fuerte, y doblegó nuestros espíritus que no pudieron detener esa innata curiosidad por casi cualquier cosa desconocida. Como todos sabéis, esto dejaba en suspenso nuestro discurrir por las vías de la ciencia, que se veían relegadas por cosas más "urgentes" o, sencillamente, más "entretenidas" cuando se comparan con la ardua labor de la escritura. Por tanto, perdóname por alargar la llegada de este desenlace, aunque bien sabes que toda idea requiere maduración, y que de cualquier cosa se aprende. Aparte de ser el

faro al que mirar para no perderme, me has enseñado a sobrevivir en el filo de las fechas límite, a optimizar el tiempo y a transferir lo aprendido en un lugar a otras muchas situaciones. Esto ha permitido que nuestros arabescos laterales terminaran, antes o después realimentando el camino principal que no deberíamos haber abandonado tan a menudo.

Carmen, tú fuiste la impulsora de esto. Como hicieras con muchos otros antes, te propusiste encauzar mi futuro hacia este mundo, y bien sabes que no fue muy difícil. Pronto pasaste el testigo a Pedro, pero es innegable que el primer empujón me lo diste tú. Aunque apuntabas ya a objetivos mucho más altos, gracias por fijarte un día en aquel par de mozalbetes que miraban a su futuro con incertidumbre, salvándonos así de dedicar nuestras vidas a dar machetazos para otros. Desde entonces, siempre has sido un eco lejano que al acercarse, cuando las obligaciones y las circunstancias lo permitían, no dejaba pasar la oportunidad de enseñarme algo.

Dentro del Departamento, muchos habéis encauzado mi viaje. Belén, has sido una fiel colaboradora, que has buscado todos los resquicios posibles por encontrar cómo facilitarme la labor, aportando por el camino también muchas ideas. Luis, me has dado todo lo necesario para poder seguir colaborando con vosotros, aun cuando las circunstancias hicieron que me subiera en otro barco. Otros habéis estado sencillamente apoyando, bien porque ya sabíais lo que era hacer este viaje, o bien porque lo estábais haciendo. Guillermo y Javier, habéis soportado algunos de mis desahogos, sirviéndome de vez en cuando como válvula de escape dando pelotazos a una pared.

A lo largo de este ya largo camino, algunos otros os habéis acercado a ratos para empujar, desde Juan Antonio que colaboraste en la programación de algunas cosas, hasta Pablo Palmier, que has sido el último en ayudar, dando un soplo de aire fresco en algunos sitios que empezaban a estancarse.

Muchos habéis servido como apoyo moral y soporte en los momentos difíciles, animándome en cada paso tambaleante, insistiendo en que no me desviara y tratando de reconducirme cuando lo hacía. Mi familia más cercana, mis padres (Pilar y Pablo) y mi hermano (*Joti*), habéis soportado estoicamente el día a día de este peregrinaje. Benjamín, gracias por inculcarme mi sed de conocimiento cuando era más joven ("y menos sabio", como tú dirías). Marisa, gracias por el apoyo de quién ha hecho ya este viaje, junto a tu buen escudero Julio.

A todos vosotros, y a aquellos que os habéis conformado con verme pasar, muchas gracias.

Y con esto, Dios te dé salud, y a mí no olvide.

Vale.

Resumen

Los juegos crean un entorno seguro en el que practicar y aprender aquellas habilidades que se necesitarán en el resto de la vida. Los *videojuegos* pueden, de la misma forma, servir como primer paso hacia la *alfabetización digital*. Los primeros juegos de arcade mejoran la coordinación ojo-mano, además de aumentar la agilidad mental o la creatividad.

Las mejoras del *hardware* han ocasionado un aumento radical en la complejidad de los videojuegos sin por ello desanimar a los jugadores, que esperan impacientes los nuevos lanzamientos. Esto ha despertado el interés de diferentes educadores deseosos de aprovechar esta capacidad de motivación para enseñar dominios más específicos, aprovechando el enriquecimiento del *lenguaje* que el medio ha conseguido.

Sin embargo, en la enseñanza de estos dominios más complejos el modelo de aprendizaje por descubrimiento (prueba y error) que los videojuegos irradian muestra sus debilidades. Se hace necesario un apoyo por parte del propio sistema para guiar al estudiante. El área de la enseñanza asistida por ordenador ha proporcionado diferentes soluciones a este problema, aunque tienden a requerir un gran esfuerzo de autoría o bien de *contenido* o bien de *conocimiento*.

En esta tesis se propone una alternativa mixta: el uso de razonamiento basado en casos intensivo en conocimiento. Los casos (*contenido*) hacen referencia a ontologías (representaciones explícitas de *conocimiento*), lo que proporciona lo mejor de ambos mundos. Los casos nos permiten enfrentarnos con cualquier dominio, y la representación del conocimiento nos permite reaprovechar una parte del esfuerzo de autoría, e incluso generar automáticamente fragmentos de las soluciones. Además, demuestra que este modelo encaja perfectamente en el ciclo normal de interacción de los videojuegos, por lo que resulta plausible integrarlo en software de entretenimiento para aprovechar la motivación intrínseca que éste produce en los usuarios.

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

Pues lo que quiero decir – dijo Sancho –, es tan verdad que mi señor Don Quijote, que está presente, no me dejará mentir.

Don Quijote de la Mancha II,
Miguel de Cervantes

Paenza (2005, página 188) plantea una situación hipotética muy curiosa, que abre la puerta a la reflexión. Imaginemos que un cirujano de principios del siglo XX, fallecido cerca de 1.920, se despertara hoy en día, y visitara el quirófano de uno de nuestros hospitales, en mitad de una operación.

¿Qué pensaría y qué haría? Claramente el cuerpo humano no ha cambiado en todos estos años, por lo que es de esperar que no debería tener ningún problema en *entender* la dolencia del paciente, o asimilar lo que le están haciendo. Sin embargo, con toda seguridad *no* comprendería las nuevas "técnicas quirúrgicas", es decir la colección de aparatos, instrumental y analíticas realizadas al enfermo. Aquí *sí hay una diferencia*: el viejo cirujano estaría *totalmente desfasado* con respecto a sus compañeros de profesión, debido al avance que la cirugía ha sufrido durante el último siglo.

Repitamos ahora este ejercicio de imaginación cambiando de profesión. En lugar con un cirujano, Paenza nos propone pensar en resucitar al maestro de primaria de nuestros abuelos o padres. Obviamente, en lugar de llevarle a un quirófano, le acompañamos a un lugar que conocía mejor: una escuela. ¿Tendría los mismos problemas de *comprensión* de las "técnicas de enseñanza"? ¿Entendería lo que está haciendo el profesor, o por qué lo hace?

Tristemente, la respuesta es que *sí*. Seguramente podría, incluso, coger la tiza y tomar las riendas de la clase sin demasiados problemas. La lectura que debemos hacer de esto es que la tecnología y el avance científico ha modificado drásticamente la forma de afrontar muchas disciplinas, pero no ha ocurrido un cambio tan radical (¡ni de lejos!) en los métodos y programas de enseñanza.

En realidad, no porque pase el tiempo las cosas tienen necesariamente que haber cambiado. Si se decide que algo *ya funciona*, y *queremos* seguir haciéndolo igual, *elegimos* no cambiar nada, entonces no hay motivo para criticarlo. Pero si el problema es que no se *evalúa* o no se *consensúa* la mejor forma de hacer avanzar una disciplina, entonces algo se está haciendo mal.

Y, a día de hoy, el sistema educativo *funciona mal*; en muchos aspectos está enquistado, enfermo de una gran inercia que mantienen vivas técnicas pedagógicas que ya no sirven.

¿Cual es la solución? En 1981, Bork aseguraba que:

> "Estamos en el preludio de una gran revolución en la enseñanza, una revolución sin precedentes desde la invención de la imprenta. El ordenador será el protagonista de esta revolución; en el año 2.000 el principal medio de aprendizaje en todos los niveles y en casi todas las áreas se realizará a través del uso interactivo de los ordenadores."

Como mostraremos en el capítulo 2, en el mundo de la Informática la educación asistida llevaba ya en 1981 muchos años investigando la mejor manera de utilizar los ordenadores para enseñar. Aunque, tras dejar atrás el icónico año 2.000, la realidad anticipada por Bork aún no se ha producido, no podemos negar que a día de hoy se percibe un cierto empuje desde las instituciones educativas por la educación asistida por ordenador.

Los cursos *on-line* comienzan a dejarse ver de vez en cuando, y los colegios e institutos han abierto, aún tímidamente, la puerta a los ordenadores. Pero, ¿es esto suficiente? Siguiendo esta línea, ¿conseguiríamos dentro de unos años que un hipotético visitante, un profesor del siglo pasado se sintiera cohibido por la nueva realidad en las técnicas de enseñanza?

En realidad, a nivel pedagógico, el uso que se da a los ordenadores continúa utilizando ideas conductistas. La formación on-line a menudo sigue dejando la gran carga del aprendizaje en la lectura y comprensión, delegando a un segundo plano el aprendizaje por descubrimiento.

Como resultado, para muchos el modelo de educación sigue siendo hoy una herencia de la cadena de montaje de Henry Ford. Bajo este punto de vista, los alumnos pasan, en serie, por cursos sucesivos donde profesores especializados les esperan para incrustar en sus cerebros conocimiento adicional a aquél que algún compañero les colocó antes. Esto constituye una cadena de producción de mentes donde los alumnos son el producto creado por los operarios, los profesores.

Es cierto que es una visión un tanto radical. Pero, se interprete como se interprete, lo cierto es que el sistema funciona cada vez peor: las *técnicas de enseñanza* han envejecido mal, y lo que sirvió durante décadas, ya no da buenos resultados hoy. Por otro lado, la visión que tienen muchos alumnos del sistema es que su objetivo es *aprobar*. Y eso no es, ni mucho menos,

sinónimo de *aprender*. La pereza despierta la picaresca, y los estudiantes buscan formas de deslizarse por el sistema consiguiendo el objetivo oficial de superar los exámenes, pero bajo la ley del mínimo esfuerzo, que suele estar reñida con el aprendizaje significativo.

Lo verdaderamente chocante de la situación se hace visible cuando se compara esta cultura del no sacrificio instaurada en nuestras aulas con el tiempo de ocio de aquellos que se sientan en sus pupitres. Muchos de ellos dedican con placer gran cantidad de horas a *labores complejas*. ¿Es sencillo meter un balón entre tres palos cuando todo un equipo contrario trata desesperadamente de impedirlo? ¿Es fácil hacer pasar un balón por un aro situado a tres metros del suelo? Claramente *no*.

Si nos vamos al *entretenimiento digital*, la situación se repite. Los videojuegos son día tras día cada vez más complicados, exigen una gran atención para ser superados y, aun así, alrededor de ellos se crean grandes comunidades de usuarios, alimentadas muchas veces por los mismos que prefieren llenar las horas matinales pensando en qué trastada hacer en lugar de aprovecharlas escuchando a su profesor.

Los deportes, juegos tradicionales y videojuegos requieren un *aprendizaje* para ser dominados. Reclaman dedicación, esfuerzo y repetición, exactamente lo mismo que se necesita para aprender a dividir, a escribir una redacción o a analizar un texto histórico. Y sin embargo, resulta irónico la opinión que tienen los alumnos. La peor propaganda para una actividad extraescolar es que sea *difícil*, mientras que para un videojuego es que sea *demasiado fácil*. ¿No sería posible *embotellar* los ingredientes de los deportes y videojuegos que consiguen su motivación, para condimentar y endulzar con ellos las amargas clases regladas?

En el capítulo 3 afrontamos esta pregunta, haciendo un recorrido por los *videojuegos educativos*. Su objetivo es mezclar la diversión del entretenimiento digital con la enseñanza. En realidad, aprender jugando es algo innato a la naturaleza humana (y a la de otros muchos animales superiores) y, de hecho, el juego y el aprendizaje son inseparables en las primeras etapas de la vida. La gran duda es si es posible llevar esta fusión a las aulas.

Curiosamente, la idea de utilizar videojuegos para enseñar estaba ya recogida en la patente que, en el lejano 1.966, hiciera Ralph Baer sobre la primera "videoconsola". A pesar de eso casi 20 años después, en 1985, Card seguía considerándolo *ciencia ficción* en su premiado libro "*El juego de Ender*".

La situación hoy sólo ha avanzado ligeramente. En general se reconoce la utilidad de los videojuegos para el aprendizaje, pero normalmente sólo en el marco infantil, como método de aproximación a las nuevas tecnologías primero, y como forma de *repasar y reforzar* el contenido que están aprendiendo en el colegio después. Normalmente, estos sistemas se desarrollan de manera manual, *sin aprovechar* el legado que el área de la enseñanza asistida ha ido dejando a lo largo del ya largo camino que lleva recorrido. En concreto, muy

pocas veces *ayudan* al alumno, y muchas menos aún le *evalúan*.

Para conseguir instaurar la costumbre de incluir en esos videojuegos educativos comerciales las técnicas que llevan tanto tiempo con nosotros en el mundo de la enseñanza asistida, es primero necesario un análisis de las diferentes formas de llevarla a cabo. En el capítulo 4 planteamos un *modelo* que organiza las ideas, y aclara la manera de incrustar un sistema de tutoría en el corazón de un videojuego. La idea de fondo consiste en fusionar un *ejercicio* (parte educativa) y un *nivel* (parte lúdica), en un claro decantamiento por el constructivismo, que encaja perfectamente en el aprendizaje por descubrimiento que irradian muchos videojuegos.

La unión de ambos mundos utilizando un nivel como engranaje principal hace encajar perfectamente el *ciclo* de ejecución de un videojuego y el de una aplicación educativa basada en problemas. No obstante, alrededor de dicho punto de unión, quedan aleteando una serie de aspectos sobre los que se deben tomar decisiones, tanto a nivel de *diseño* del videojuego (género, historia de fondo, etcétera) como del *sistema de tutoría*. Por ejemplo, ¿cómo elije el programa el siguiente ejercicio a resolver? ¿Cómo se puede analizar la *corrección* de su solución? ¿Cómo y cuándo se proporciona ayuda al jugador? Tradicionalmente estas respuestas se han contestado *simultáneamente* en los diferentes sistemas educativos. Nosotros, sin embargo, desgajamos cada una de ellas colocándola en una etapa diferente de los mencionados "ciclos" de ejecución y comprobamos que, de hecho, las decisiones sobre el modo de solventar cada una de ellas pueden tomarse de manera relativamente independiente.

Con todo lo anterior, y por resumir, el mencionado capítulo 4 describe un *espacio de implementación*, especialmente en lo referente a la parte pedagógica de los videojuegos educativos con varios *ejes* en los que tomar decisiones. Por ejemplo, la *elección* del siguiente ejercicio (o nivel) al que enfrentar al jugador se puede hacer principalmente de dos maneras. La primera se basa en la *generación* del ejercicio, a partir de *conocimiento* del dominio con el que dotamos al sistema de la capacidad de *inventar al vuelo* tareas que cumplan un determinado objetivo pedagógico. La segunda opción se basa en la *selección* del ejercicio, extrayéndolo de una *base de problemas* creada de manera manual.

Por otro lado, el sistema necesita conocer la *solución* del ejercicio para cotejar la proporcionada por el usuario dentro del juego. Esta solución también puede conseguirse de varias maneras que, en principio, son *independientes* del modo en el que se haya conseguido el ejercicio. Partiendo *únicamente* del enunciado, recogido de la etapa anterior, podemos crear la solución, de nuevo, con *otro* sistema experto, o con *otra* base de casos de soluciones.

Con esto se quiere dar la idea, mucho más detallada en su correspondiente capítulo, de que, efectivamente, se dispone de un *espacio de alternativas* a

la hora de desarrollar un videojuego educativo. Este análisis puede utilizarse como inspiración para las fases iniciales del diseño de un videojuego concreto, permitiendo aclarar ideas y encontrar su posición dentro del marco general dibujado por el modelo.

Naturalmente, las decisiones en cada eje dependerán enormemente del dominio que se pretende enseñar. Conseguir que el ordenador se "invente" ejercicios de operaciones aritméticas es mucho más sencillo que lograrlo para ejercicios de física. Así, cuando el dominio adquiere cierta complejidad, el uso de sistemas expertos en la fase de elección del ejercicio se hace increíblemente complicada. Los sistemas *basados en casos* empiezan por ello a mirarse con buenos ojos, aunque no puede olvidarse el problema de *autoría manual de ejercicios* que traen consigo.

Como alternativa, nosotros nos decantamos por el uso de *razonamiento basado en casos rico en conocimiento*. Esto decide, en cierto modo, quedarse en el punto medio entre los casos y las reglas, al enriquecer los primeros con conocimiento adicional que *se comparte* entre todos ellos, aliviando el coste de la autoría. En concreto, trata de *hacer explícito* el conocimiento general sobre el dominio, que de otra manera quedaría incrustado (y, peor aún, *desaprovechado*) dentro de los casos.

Además, en nuestro modelo de aplicaciones educativas en las que, en realidad, hay varios *subsistemas independientes*, la extracción de ese conocimiento añade ventajas de reutilización adicionales. Por ejemplo, el conocimiento general extraído del sistema de *selección* de ejercicios puede reaprovecharse en el que se encarga de la *resolución*. Aparece así una reutilización *entre subsistemas*, que va un paso más allá de la reutilización *entre casos*.

Para poner a prueba todas estas ideas, los capítulos 5 y 6 plantean dos instanciaciones *diferentes* del modelo de integración de videojuegos y sistemas educativos que comentábamos antes. En realidad ambos *enseñan el mismo dominio*, dado que resultaba mucho más cómodo al desligarnos de la necesidad de expertos externos y facilitaba la comparación entre ambos. Esto *no* significa que nuestro modelo general esté *atado* a dicho dominio; sencillamente era más cómodo para nosotros.

Ambos sistemas son lo suficientemente diferentes como para que cada uno explore una pequeña región del espacio de alternativas del que hablábamos antes. La primera instanciación de nuestro sistema toma la decisión de utilizar técnicas de razonamiento basado en casos relativamente primitivas, lo que sirve como una buena prueba de concepto del modelo general, pero sufre problemas en el *coste de autoría* exigiendo un gran trabajo de los expertos humanos que deben alimentar el programa. En lo referente a los aspectos lúdicos, integra un juego *de aventura* con un modo de interacción muy conocido en el mundo de los videojuegos, lo que alivia la curva de aprendizaje del modelo de interacción. Además, posee una historia de fondo que justifica la componente pedagógica y crea un marco en el que encajan los diferentes

personajes.

La segunda instanciación se desplaza hacia técnicas de razonamiento más académicas. En concreto, el razonamiento basado en casos *rico en conocimiento* brilla aquí con especial esplendor, sirviendo como demostración de sus bondades, que ya hemos mencionado. Las labores de *autoría* del contenido educativo se suavizan gracias a la extracción de este conocimiento general (en forma de ontologías), que es utilizado a lo largo y ancho de los diferentes subsistemas en los que se descompone la aplicación. Esto aporta también ventajas adicionales al análisis que el sistema es capaz de realizar de las acciones del usuario, pudiendo extraer conclusiones más acertadas sobre lo que éste *hace mal*.

Otro punto de diferencia respecto a la primera versión lo encontramos en el *género del videojuego* que sirve de sustrato a la componente educativa. En concreto, se ha transformado en un juego de *acción*, donde las habilidades de manejo de ratón y teclado resultan más exigentes, pero a la vez añade dinamismo y movimiento al programa. Esto incita al usuario a *agilizar* la resolución de los ejercicios lo que supone una *integración intrínseca* positiva al imponer, desde la componente de juego, un requisito de velocidad que también genera una presión interesante en la parte pedagógica.

1.1. Publicaciones

Este trabajo ha originado diversas publicaciones a lo largo del tiempo. Aunque se encuentran referenciadas en los momentos en los que se menciona su contenido, se describen aquí brevemente las más significativas.

- En Gómez Martín et al. (2004a) y González Calero et al. (2006) describimos las bondades de los videojuegos educativos de manera general.
- La descripción de las cinco etapas de ejecución de los tutores inteligentes (selección de conceptos, ejercicios, etcétera) está descrita en Gómez Martín et al. (2005a,b).
- La idea de fusión de un videojuego y un tutor inteligente que sigue la enseñanza basada en ejercicios está descrita en Gómez Martín et al. (2004b).
- La primera instanciación del modelo de videojuegos educativos (JV^2M) está descrita, con diferente grado de detalle y en diferentes estados de evolución, en Gómez Martín et al. (2003, 2005d, 2007a).
- La segunda instanciación del modelo (JV^2M 2) se describe, a nivel de diseño de videojuego, en Gómez Martín et al. (2006a,c).

- En Gómez Martín et al. (2007b) se comparan los diseños (metáforas) de JV^{2}M y JV^{2}M 2.
- En Gómez Martín et al. (2005c) se describe el funcionamiento interno de JV^{2}M 2 a nivel de los ejercicios presentados al usuario.

Capítulo 2

Aprendizaje asistido por ordenador

– ¡Ah pecador de mí –respondió don Quijote–, y qué mal parece en los gobernadores el no saber leer ni escribir! [...] Gran falta es la que llevas contigo, y, así, querría que aprendieses a firmar siquiera.

Don Quijote de la Mancha, Miguel de Cervantes

Resumen: En este capítulo se hace un recorrido por las diferentes familias de programas educativos que han ido surgiendo a lo largo de los años. Para ponerlos en contexto y poder comparar sus diferentes aproximaciones, se comienza realizando una descripción de la importancia de la educación, y de las diferentes teorías pedagógicas que los siglos nos han dejado. Después se describen los sistemas, desde los primeros que se basaban en marcos hasta los más modernos agentes pedagógicos.

2.1. Introducción

La educación es uno de los valores más importantes que la civilización proporciona a sus ciudadanos. La cantidad de dinero dedicada a ella atestigua el poder adquisitivo de cada país, y es una de las medidas más significativas de su *capital humano*.

No es de extrañar, por lo tanto, que durante siglos se haya estudiado con cuidado el modo en el que los humanos aprendemos, con la intención de mejorar el proceso de la educación. Desde el aprendizaje condicionado de los famosos perros de Pavlov hasta las teorías más recientes del constructivismo

han intentado explicar el funcionamiento profundo de nuestros pensamientos. Psicología y pedagogía han ido de la mano durante todos estos siglos, una buscando respuestas a este modo de pensar, y la otra aprovechándolas para optimizar el sistema educativo.

Con la llegada de los ordenadores, surgió rápidamente la idea de aplicar esas teorías a *programas educativos* que solventaran las carencias de la enseñanza en clases multitudinarias. Como en muchas otras áreas, fueron las necesidades militares las que impulsaron la investigación en este campo. Las grandes exigencias de formación y las carencias de formadores hicieron que, en tiempos de guerra, muchos miraran con otros ojos a las incipientes aplicaciones de enseñanza. La posibilidad, además, de proporcionar *instrucción sin riesgo de pérdidas humanas* también empujaron a la creación de los cada vez más sofisticados simuladores.

Este capítulo hace inicialmente un recorrido rápido por las ventajas de la educación, así como por las diferentes teorías pedagógicas que han ido apareciendo con el tiempo. Después se adentra en la aplicación de estas teorías en las distintas familias de programas educativos que nos ha deparado el mundo de la informática.

2.2. La importancia de la educación

En el siglo XII, el filósofo francés Bernard de Chartres utilizó una metáfora (hoy a menudo incorrectamente atribuida a Isaac Newton) sobre el saber humano. Dijo que "somos como enanos sentados sobre los hombros de gigantes, que ven más, y a mayor distancia, de lo que vieron nuestros antepasados". Esta capacidad de llegar más allá de lo que alcanzaron los que nos precedieron no se debe a una mayor capacidad nuestra, sino a que nos apoyamos en lo que otros hicieron antes de que llegáramos[1].

Y es que nuestra sociedad se mantiene sobre los hombros de los genios que nos precedieron. El conocimiento acumulado a lo largo de los siglos nos hace ver como algo natural las comodidades que hoy tenemos, incluyendo cosas "obvias" como el agua corriente, el lápiz y papel o la luz eléctrica, o logros más complejos tales como la televisión, Internet, o los *scanners* médicos.

Para poder seguir evolucionando es necesario *comprender* el punto al que hemos llegado y dar un paso más allá; apoyarse en los hallazgos previos y continuar, despacio pero sin pausa, hacia un futuro con más conocimientos. Sólo siguiendo ese camino se hará cierta la conocida creencia de que "las generaciones futuras se reirán de lo poco que sabemos".

Para alcanzar ese objetivo es necesario que los miembros de la sociedad dispongan del conocimiento que nos han dejado como legado nuestros ante-

[1] Como muestra de la fama de esta frase, baste decir que una versión reducida en inglés (*"Standing on the shoulders of giants"*) está grabada en el lateral de las monedas de dos libras esterlinas.

pasados. Para eso, es imprescindible *la educación*, que permite la transmisión de dicho conocimiento, junto con las destrezas y los valores, que la humanidad ha ido acumulando durante toda su existencia. De esa forma, tal y como dice Bennett (1999), los que nos sucedan podrán desarrollar sus propias capacidades, e integrarlas en sus vidas. Mantener y propagar el legado histórico y cultural es por lo tanto de vital importancia para el futuro. De ahí que sea innegable la importancia de los centros religiosos de la Edad Media, que tercamente se empeñaron en mantener los conocimientos conseguidos por sus ancestros a pesar de vivir en una época difícil y oscura.

Todo esto convierte a la educación en un factor clave para la evolución de la sociedad. A priori no es posible averiguar quién será el próximo Edison, o el futuro Mozart. Lo que sí está claro es que para conseguir descubrirlos, necesitan *ser educados*, por lo que habrá más probabilidad de encontrarlos si no desperdiciamos oportunidades dejando a potenciales genios sin instrucción. Por tanto, parece claro que maximizar el avance de la civilización pasa por educar a todo el mundo, para potenciar sus habilidades innatas y sacar el máximo partido de ellas. Esto desemboca en la idea general de que la educación (y, en particular, la educación superior) es un indicador muy poderoso sobre el *capital humano* de un determinado país.

Desafortunadamente, la globalización en la educación no es trivial. En primer lugar hay muchas posibles formas de habilidad, y encontrar la que más se adapta a cada uno no siempre es fácil. Después de todo, puede que alguno de nosotros tengamos una facilidad natural para la búsqueda de oro, pero nunca se nos dio la oportunidad de ponerla a prueba. La solución habitual es limitarse a enseñar una serie de materias que se consideran lo suficientemente generales y útiles como para que merezcan la pena ser conocidas por todo el mundo, con la esperanza de que cada cual encuentre su destino como mejor pueda.

En segundo lugar (y quizá más importante), la enseñanza (y en general, la cultura) *es un lujo*. Como muestra, baste decir, por ejemplo, que la filosofía sólo surgió cuando la gente comenzó a tener el resto de necesidades básicas cubiertas, es decir, cuando las posibilidades económicas fueron propicias. Un ejemplo quizá más cercano es el comentado por Peña Marí (2006). Tanto Charles Babbage como Augusta Ada Lovelace (dos nombres ilustres de la prehistoria de la Informática) fueron personas adineradas que tuvieron la suerte de nacer en familias de la alta sociedad con grandes recursos. Sólo disponiendo de dichos recursos es posible permitirse el lujo de recibir educación pues requiere mucho tiempo y tarda en dar sus frutos.

Hoy en día, los gobiernos son conscientes de que el bienestar económico de un país depende en gran medida de la actividad económica y laboral de su población. Ésta, a su vez, resulta fuertemente dependiente de la preparación del capital humano conseguida con su educación. Independientemente de esta visión, quizá egoísta al estar pensada para la mejora de la *sociedad* a medio

plazo, también es claro que la instrucción es beneficiosa para *el individuo*. Esto hace que la educación sea considerada *un derecho*, tal y como plasmó la ONU (Organización de las Naciones Unidas) en 1.948 en la Declaración Universal de los Derechos Humanos[2] o mucho después, en 1.989, se hiciera en la Convención sobre los Derechos del Niño. De hecho, su resolución principal adquirió, tan sólo un año después, el carácter de Ley Internacional de modo que los gobiernos están *obligados* a impulsar y proporcionar la educación *primaria* de forma gratuita para todos.

2.3. La educación en cifras

No cabe duda de que la posibilidad de ofertar esa educación gratuita depende del bienestar económico; sólo los países con un PIB (Producto Interior Bruto)[3] razonablemente abultado pueden permitirse el lujo de dedicar más dinero a la educación, por ejemplo, proporcionando de forma gratuita formación más allá de la educación primaria, o facilitando estudios superiores. De nuevo, la razón vuelve a ser la misma. En general la educación supone mantener a niños y adolescentes *que no producen*, como una apuesta de futuro.

Normalmente se considera educación *primaria* la que comienza entre los 6 y 7 años, y termina entre los 11 y 12. Resulta significativo el informe conjunto realizado en 2005 por UNESCO y UNICEF[4]. Estima que, en un mundo que promete educación primaria universal, hubo aproximadamente 115 millones de niños sin escolarizar durante el curso 2.001/02, lo que constituye el 18 % (casi uno de cada cinco) de los niños del mundo. Es bastante significativo observar que de todos ellos, el 39 % residían en África, y el 36 % en la parte Sur de Asia, mientras que sólo el 2 % lo hacía en los países industrializados[5]. En total, el 82 % de los niños sin escolarizar habitan en zonas rurales. Por último, considerando únicamente las familias donde hay niños sin escolarizar, es muy representativa la conclusión de que aquellos que viven en el 20 % de los hogares más pobres tienen el triple de probabilidades de no ser escolarizados al compararlos con los niños que habitan en el 20 % de los hogares más ricos.

Para centrarnos en lo que ocurre en nuestro país, podemos analizar las

[2] `http://www.un.org/Overview/rights.html`

[3] El PIB de un país constituye una medida económica que indica el valor de los bienes y servicios producidos en el territorio nacional durante un periodo. Según Wikipedia (Producto_interior_bruto-es), el PIB es, sin duda, la macro-magnitud económica más importante para la estimación de la capacidad productiva de una economía.

[4] La UNESCO (*United Nations Educational, Scientific and Cultural Organization*, Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura) nació el 16 de Noviembre de 1.945 con un objetivo amplio y ambicioso: construir la paz en la mente de los hombres mediante la educación, la cultura, las ciencias naturales y sociales y la comunicación (`http://www.unesco.org`).

[5] Se entiende por países industrializados a gran parte de Europa, a Estados Unidos y Canadá, y algunos otros países y territorios como Japón y Hong Kong.

cifras publicadas en 2.005 y 2.006 tanto por el Ministerio de Educación y Ciencia como por el INECSE (Instituto Nacional de Evaluación y Calidad del Sistema Educativo)[6].

En total, en 2.003 se gastaron en España casi 34 millones de euros públicos, de los cuales poco más de dos millones y medio se destinaron a becas. No obstante, las cifras absolutas no son de gran ayuda, pues no permiten hacerse una idea de su magnitud al no tener ningún otro dato con el que compararlas. Es mucho más interesante el porcentaje del PIB que se destina a educación, lo que da una idea de la importancia que para el gobierno tiene la instrucción de los ciudadanos. En 2.002, España gastó en educación el 5'6 % de su PIB, repartido entre el 4'5 % a costa de las arcas del estado, y el 1'1 % en financiación privada (generalmente por parte de las propias familias). En realidad, tampoco este porcentaje da una idea completa, pues es muy dependiente del valor concreto del PIB, de la proporción de población en edad escolarizable, de las tasas de escolarización y de la propia estructura del sistema educativo. Por ejemplo, diez años antes, en 1.992, el porcentaje del PIB destinado a educación era superior (el 5'9 %), pero en contraposición el valor del PIB era mucho menor y la población escolarizada sensiblemente mayor. El resultado es que en 1.992 el gasto absoluto en educación fue prácticamente la mitad de lo que fue en 2.002. En este sentido, quizá resulte también clarificador comparar el gasto medio por alumno escolarizado, que en 1.992 fue de 1.640 Euros, mientras que en 2.001 ascendió a 3.900[7].

Por otro lado, aunque sea teniendo en mente que, como se ha dicho, el porcentaje del PIB destinado a educación en realidad es únicamente un indicador parcial, es posible utilizarlo para comparar el gasto en educación entre España y otros países de la Unión Europea. En este sentido, es significativo que el porcentaje destinado de nuestro dinero público (un 4'5 %) esté por debajo del de la media de la Unión de los 15[8] (un 4'9 %) sólo por encima de Grecia e Irlanda.

Por último, y aunque sea también una medida indirecta del coste de la educación, las estadísticas muestran cierta relación entre la situación laboral del padre[9] y la posibilidad de estudio de los hijos. En 2.003, los porcenta-

[6]Esta es la nueva denominación del antiguo INCE (Instituto Nacional de Calidad y Evaluación) establecida por la LOCE (Ley Orgánica de Calidad de la Educación). Su página web está disponible en `http://www.institutodeevaluacion.mec.es/`

[7]Naturalmente, estos datos se han convertido desde su valor original medido en pesetas. Además, de manera anecdótica, en realidad el valor expresado aglutina todos los niveles educativos. En 2.001, se gastó por niño escolarizado *en educación infantil* una media de 2.742 euros, mientras que por estudiante de educación superior fue más del doble, 5.566.

[8]Dado que las cifras son anteriores, no se tienen en cuenta ni los 10 países que se incorporaron a la Unión en Mayo de 2.004 ni los 2 adheridos en 2.007. Sólo se consideran, por lo tanto, los datos de Alemania, Austria, Bélgica, Dinamarca, España, Finlandia, Francia, Grecia, Irlanda, Italia, Luxemburgo, Países Bajos, Portugal, Reino Unido y Suecia.

[9]El INECSE no hace explícito a qué se refiere con el apelativo de "padre". A pesar de que la mayor parte de los análisis estadísticos que realizan proporcionan información por

jes de jóvenes que tenían posibilidad de acceso a estudios universitarios se ordenaban de menor a mayor entre jóvenes con padre parado (33 %), seguidos por los jóvenes con padre inactivo[10] (40 %) y por último de los jóvenes con padre ocupado (49 %). Estas cifras, aparte de resultar significativas en nuestra discusión sobre el coste de la educación, lanzan, quizá, una señal de alarma respecto a la calidad del sistema de becas para ese tipo de estudios.

Las estadísticas anteriores vienen a demostrar el hecho mencionado inicialmente de que la educación *es cara*. Una cuestión que intuitivamente es obvia pero que aún queda por demostrar es si la educación *es útil*. Es decir, ¿realmente merece tanto esfuerzo la tarea de educar?

Para contestar a esa pregunta tenemos que ver si el resultado a largo plazo de la educación merece la pena, tanto a nivel general sobre la mejora de la sociedad y la bonanza del país, como a nivel individual sobre la situación particular de cada persona.

En relación al ámbito global, podemos analizar la repercusión que tiene la educación en la posibilidad de encontrar un puesto de trabajo. Esto se realizará bajo la hipótesis de que a mayor número de trabajadores, mejor beneficio recibirá la sociedad de la inversión que hizo años antes educando a sus niños y jóvenes.

Por otro lado, para estimar los beneficios individuales podemos preguntarnos si realmente la educación resulta fundamental para conseguir un puesto de trabajo con *mejores condiciones laborales*.

Antes de adentrarnos en los resultados de este tipo de estudios merece la pena detenerse a recordar la definición de algunos términos que han surgido previamente y que volveremos a utilizar en breve[11].

- *Población total*: población de 16 y más años que habita en viviendas familiares.

- *Población activa*: personas de 16 o más años que suministran mano de obra para la producción de bienes y servicios o están disponibles y en condiciones de incorporarse a dicha producción. Se subdividen en ocupados o parados.

    - *Población ocupada*: Personas de 16 o más años que, durante la semana de referencia, han estado trabajando durante al menos una hora, a cambio de una retribución (salario, jornal, beneficio empresarial, etc.) en dinero o en especie. También son ocupados

género, en este caso no aclaran si se refieren exactamente al progenitor masculino de los estudiantes o, quizá, al de mayor nivel de estudios de ambos.

[10] En seguida recordaremos la diferencia entre una persona parada y una inactiva.

[11] Estas definiciones son las utilizadas por el INE (Instituto Nacional de Estadística) en la EPA (Encuesta de Población Activa)

quienes teniendo trabajo han estado temporalmente ausentes del mismo por enfermedad, vacaciones, etc.

- *Población parada (desempleada)*: Personas de 16 o más años que, durante la semana de referencia, han estado sin trabajo, disponibles para trabajar y buscando activamente empleo. Son parados también quienes ya han encontrado un trabajo y están a la espera de incorporarse a él, siempre que verifiquen las dos primeras condiciones.

- *Población inactiva*: población de 16 o más años no incluida en las categorías de población activa o población contada aparte.

- *Población contada aparte*: los varones que cumplen el servicio militar[12].

En 2.002, a instancias de un reglamento de la Unión Europea, se modificó la definición de "buscando activamente empleo". Para que alguien pudiera entrar en esa categoría, debía haberse puesto en contacto con su oficina del INEM (Instituto de Empleo) en algún momento durante las cuatro semanas anteriores a la toma de las muestras. Eso hizo disminuir el número de parados con gente que pasó a engrosar la lista de población inactiva.

Una vez definidos estos conceptos, retomamos la cuestión sobre si la educación beneficia realmente a la sociedad. Asumiendo que, efectivamente, un alto número de trabajadores es positivo, nos interesa conocer estadísticas que nos informen de la relación entre el nivel educativo y la situación laboral.

En ese sentido, el INECSE ha publicado un informe con cifras del año 2.003 que indican que, a mayor nivel de estudios, también es mayor el porcentaje de actividad. Así, el 88'1 % de la población con estudios superiores se encuentran dentro de la *población activa* (trabajando o en disposición de hacerlo). Este porcentaje baja al 80 % para las personas con estudios postobligatorios, al 75 % con estudios secundarios, y se queda tan sólo en el 53'3 % en el caso de gente con estudios primarios o inferiores.

Este dato es bastante representativo, pero no es completo. Como complemento, también ha publicado, para ese mismo año, la tasa de *desempleo* según el nivel educativo. De nuevo, resulta significativo que la tasa de desempleo entre la población con estudios superiores esté por debajo de la media total, mientras que para aquellos que tienen estudios de secundaria o inferiores queda por encima.

Si lo anterior viene a demostrar que, efectivamente, el *capital humano* mejora con la educación, aun queda pendiente la pregunta de si a nivel individual, la educación merece la pena, es decir si resulta fundamental para conseguir un puesto de trabajo con mejores condiciones laborales. Aunque

[12] A día de hoy este grupo de población ha perdido el sentido.

resulte quizá un poco egoísta (y en cualquier caso sesgado), en la práctica el factor más sencillo de medir dentro de las mencionadas condiciones laborales es el sueldo conseguido por el empleado. De nuevo el INECSE viene en nuestra ayuda, con su indicador que mide las diferencias de ingresos laborales según el nivel de estudios.

Para no utilizar cifras absolutas en salarios, tomamos como referencia el ingreso medio conseguido por la población con estudios secundarios post-obligatorios, considerándolo así el 100 %. Con esta referencia, en 2.000 el salario medio de la población con estudios inferiores a los obligatorios fue tan sólo del 38 %, mientras que para los poseedores de títulos universitarios alcanzaba el 192 %. Esto demuestra que, efectivamente, el logro de niveles de estudios más altos contribuye a formar ciudadanos más cualificados, con mayores posibilidades de promoción e ingresos económicos más elevados.

Los datos anteriores muestran los aspectos positivos de la educación al constatar que a mayor nivel, mayor probabilidad hay de obtener un trabajo bien remunerado. También puede analizarse la situación inversa, buscando indicios de que la educación está relacionada con el *índice de criminalidad*. En este sentido, Edmark (2005) encontró que entre los años 1.988 y 1.999, Suecia sufrió un incremento en el número de delitos (hurtos, y robos de coches y bicicletas), coincidente con un alarmante aumento del desempleo. A nivel nacional, Rodríguez Andrés (2003) mostró que la educación era un factor relacionado con la criminalidad. Obviamente, todos estos resultados hay que tomarlos con mucha cautela. Por un lado existen una gran cantidad de variables muy relacionadas, por lo que esta afirmación de que a menos nivel de estudios mayor criminalidad debe ser siempre mirada con recelo. Además, incluso aunque fuera cierta, es muy fácil desvirtuar su significado, pudiendo avivar sentimientos xenófobos en gente de ideas radicales bajo la protección de una estadística dudosa.

Para finalizar, tan sólo una curiosidad adicional. Las estadísticas demuestran que aparte de las ventajas que la educación (especialmente la superior) tiene sobre sus receptores, parece que además ocasiona un efecto en espiral sobre las generaciones futuras. En concreto, el INECSE también publicó un estudio sobre las expectativas del nivel máximo de estudios que los padres tienen de sus hijos (y éstos de sí mismos) en función de la formación del padre. En general, en nuestro país se constata que a mayor nivel de estudios de los padres, mayor es el nivel de estudios que quieren que terminen sus hijos, y mayor es también el nivel de estudios que ellos mismos querrían terminar. Esto es algo más que anecdótico si asumimos la creencia común de que el rendimiento escolar está fuertemente influido por las expectativas que el alumno tiene del nivel de estudios al que quiere llegar (que, a su vez, suele también depender de la de los padres).

No obstante, como los deseos pueden estar lejos de lo que luego termina ocurriendo en la realidad, un último estudio comprobó que, efectivamente,

en 2.003 la probabilidad de que un joven tuviera los estudios suficientes para acceder a la Universidad era mayor cuanto mayor fuera el nivel educativo del padre. Esto demostró que, después de todo, las esperanzas de los alumnos a menudo se cumplían.

A nivel mundial, la situación es parecida. Según el informe conjunto de UNESCO y UNICEF (2005), los niños nacidos de madres sin estudios primarios tienen más del doble de probabilidad de no recibir educación alguna que los nacidos de madres con algún tipo de estudios. En realidad, este resultado también debe verse condicionado por el coste de la educación y la probabilidad de alcanzar un puesto de trabajo gracias a un mayor nivel educativo tal y como se ha comentado ya, por lo que debe considerarse con cautela.

2.4. La educación y las teorías del aprendizaje

Todo lo anterior demuestra que efectivamente, la educación es útil. Conseguir una educación superior tiene efectos positivos en la vida laboral de una persona lo que, a su vez, repercute en la economía del país. Por tanto, generalmente la educación parece una buena inversión. Pero, como ocurre con toda inversión, es necesario tratar de *maximizar* su rentabilidad, es decir conseguir que, efectivamente, cada euro aportado se aproveche lo mejor posible. Debido a esto, el fracaso escolar es visto en todas partes como un problema grave que hay que tratar de paliar, pues supone dedicar dinero a gran cantidad de alumnos que desaprovechan la oportunidad de aprender.

Por desgracia, en la comunidad educativa es una práctica habitual considerar a los alumnos los culpables de su propio fracaso. No obstante, el ser humano tiene una innata curiosidad que le incita a tratar de aprender: cuando se les da la oportunidad, todos los niños aprenden. Culpar a los estudiantes permite a los profesores mantener limpia su conciencia, pero en general no está probado que entre los alumnos con fracaso escolar la mayoría tengan incapacidad para el aprendizaje. Esto sitúa a la comunidad educativa en la desagradable situación de ser la responsable de buena parte de su fracaso, al no ser capaz de hacer una enseñanza lo suficientemente entretenida como para que los estudiantes la encuentren atractiva.

Disponer, por tanto, de buenos sistemas educativos es de vital importancia para la sociedad. Pedagogos y psicólogos han buscado los mejores métodos de enseñanza, analizando el modo en el que el ser humano aprende. En este sentido, se han planteado diferentes teorías educativas que intentan entender los procesos que originan el aprendizaje para, una vez comprendidos, diseñar métodos de enseñanza efectivos. De esta forma, cada teoría educativa suele tener asociada un método pedagógico.

Antes de adentrarnos en las alternativas más representativas que nos ha legado la historia, replanteémonos el significado de la palabra *educación*. Según Vázquez Gómez (2003), la educación "es una acción y un proceso

intencional, continuo y sistemático de perfeccionamiento de la persona en cualquiera de sus dimensiones (intelectual, física, estética, social, profesional, ética, ...)".

El adjetivo *continuo* no es casual. La educación *no* puede restringirse a los años de escolaridad, sino que dura toda la vida. En este sentido, no fue hasta 1974 cuando Coombs y Ahmed hicieron explícita la diferencia entre la educación *formal* y *no formal*:

- *Educación formal*: es la educación proporcionada por el *sistema educativo*, regido por las decisiones institucionales, organizado cronológicamente y de manera jerárquica. Abarca desde el primer año de la escuela primaria hasta el último de Universidad.

- *Educación no formal*: es toda actividad *organizada y sistemática* cuyo fin es la educación, pero que se realiza *fuera* del ámbito de la educación formal (el sistema educativo oficial), para facilitar el aprendizaje en áreas específicas o a grupos particulares de población.

La existencia de la educación no formal tiene sentido, ya que, como se ha dicho, la educación es un proceso que dura *toda la vida*, y, obviamente, la educación formal no puede alargarse indefinidamente.

En realidad, la única diferencia entre educación formal y no formal es *la acreditación* (el título oficial), ya que ambas están organizadas, sistematizadas y enfocadas a una población bien definida. En este sentido es fácil confundir la educación no formal con la a menudo conocida como *educación informal*, que es aquella actividad que genera efectos educativos pero carece de una estructura y organización que determine exactamente los objetivos que persigue.

Curiosamente, una misma actividad puede utilizarse en los tres "tipos" de educación:

- *Educación informal*: un grupo de alumnos que ha ido a la Facultad de Informática de la Universidad Complutense de Madrid a informarse sobre las titulaciones ofertadas, termina paseando por el Museo de Informática García Santesmases[13] y leyendo la información allí expuesta.

- *Educación no formal*: un grupo de personas asiste a la cena anual organizada por la Asociación de Antiguos Alumnos de esa misma Facultad[14]. José Manuel Mendias, responsable del museo, ofrece una visita guiada en la que explica las piezas más representativas, contando anécdotas interesantes sobre su historia.

[13] `http://www.fdi.ucm.es/migs/`
[14] `http://antiguosalumnos.fdi.ucm.es`

- *Educación formal*: un grupo de alumnos de un Ciclo de Informática de Formación Profesional realiza para una de sus clases un trabajo sobre la historia de la Informática. Tras realizar una investigación en libros e Internet, realizan con su profesor una visita al museo en el que pueden ver con sus propios ojos parte de los dispositivos sobre los que han encontrado información.

Esta separación en los tres tipos de educación tiene cierta relación con la distinción entre *contenido lineal* y *contenido dinámico* que Aldrich (2005) realiza en los capítulos 6 y 7 de su libro. Llama contenido lineal a aquel en el que hay un claro *eje de evolución* o de *nivel de aprendizaje*. En este eje, podríamos poner en orden el conocimiento de los alumnos. Este tipo de contenidos se utiliza en lo que denominábamos educación formal y no formal, y se aplica, naturalmente, en las clases presenciales tradicionales, pero también en los libros de texto o en los vídeos educativos.

Por otro lado, el contenido dinámico es aquel en el que no existe un orden claro o, dicho de otro modo, ordenar a un grupo de gente de acuerdo a su conocimiento en ese ámbito pierde sentido. El aprendizaje informal, con su "contenido" dinámico y divergente encaja claramente dentro de este grupo. Por ejemplo, el aprendizaje con un mentor a menudo resulta muy desordenado si éste no ha planeado con cuidado los casos con los que enfrentar a su pupilo, o el aprendizaje de un determinado tema realizando búsquedas en Internet ocasiona que cada individuo adquiera una visión diferente (y quizá sesgada) de él.

En general, dice Aldrich, el aprendizaje con contenido dinámico resulta más caro e impredecible por culpa de su falta de estructura, por lo que su aplicación a un grupo grande resulta arriesgada.

La separación entre educación formal, no formal e informal ataca la primera parte de la definición de educación dada anteriormente. También se mencionaba que la educación pretende el perfeccionamiento de la persona *en cualquiera de sus dimensiones*. Según Castillejo Brull et al. (1993), cada una de las dimensiones de la educación es la extensión que toma la educación en una dirección determinada. Normalmente, se piensa que las enseñanzas regladas están enfocadas principalmente a la *dimensión intelectual* de la educación o, dicho de otro modo, a proporcionar conocimientos científicos, técnicos, humanísticos o históricos (transmisión cultural). También suele incluirse dentro de esta dimensión intelectual la *formación intelectual*, que busca que los alumnos adquieran hábitos intelectuales, técnicas de trabajo y búsqueda de información o, más llanamente, *enseñar a pensar*.

Esto es cierto sobre todo en la educación superior. Pero los sistemas educativos hoy en día ponen especial énfasis en otras *dimensiones*, de modo que se dice que los profesores (especialmente de primaria y secundaria) no tienen como papel principal el de *formadores*, sino el de *educadores*, atendiendo así al resto de dimensiones como la social, moral y afectiva. Una muestra clara

de este hecho lo tenemos en nuestro propio país en la denominación oficial de los Institutos de *Educación* Secundaria (IES) en lugar de Institutos de *Enseñanza* Secundaria.

El análisis anterior nos ha permitido aclarar el significado de la educación. Pero, ¿cómo conseguimos inculcar en un alumno aquello que queremos?. Para esta labor, llegan en nuestra ayuda las *teorías sobre el aprendizaje*[15]. Al ser el aprendizaje una actividad compleja, no existe una única teoría definitoria que explique su funcionamiento hasta el más mínimo detalle, y, de hecho, históricamente han ido surgiendo diferentes opiniones que han ocasionado distintas teorías para explicarlo. Normalmente, cada una de estas teorías propone un método de enseñanza, es decir técnicas para conseguir que el aprendizaje tenga lugar. En las siguientes secciones veremos las dos tendencias más representativas.

2.4.1. Conductismo

Si olvidamos la parte *racional* que nos caracteriza, podemos considerar al aprendizaje como un *instrumento* para resolver los problemas y dificultades que el medio nos presenta. Esto es explicable de manera global incluso con las teorías evolutivas de Darwin. Cuando se encuentra un modo de solucionar un problema, se realiza una asociación entre éste y su solución, que volverá a utilizarse si surge de nuevo el problema que soluciona: se ha producido aprendizaje. Este modo de resolver problemas es, curiosamente, el que inspira la rama de la Inteligencia Artificial conocida como *razonamiento basado en casos* que utilizaremos en diferentes puntos a partir del capítulo 4.

Bajo esta definición de aprendizaje, el conductismo considera que los conocimientos poseídos son percibidos *mediante la conducta* (de ahí el nombre), considerada como la manifestación externa de los procesos mentales internos, que no son comprendidos. La efectividad de la enseñanza puede así ser medida a partir de los resultados conseguidos por los alumnos ante pruebas a las que se les enfrenta tras el periodo de aprendizaje.

El conductismo simplifica el aprendizaje a *reacciones condicionadas*, buscando conseguir que los alumnos, ante un determinado estímulo, generen una respuesta concreta. Los primeros estudios en este sentido los realizó Pavlov[16], cuyos experimentos sobre la salivación de los perros son universalmente conocidos.

No obstante, quien acuñó el término del conductismo fue Watson en su artículo publicado en 1913 que supuso un hito en la historia de la psicología. Él fue el impulsor de dicha rama en las teorías del aprendizaje, que persigue

[15] En general, se entiende por *aprendizaje* al proceso a través del cual se produce la educación. El aprendizaje ocasiona un cambio permanente de la persona mediante la adquisición de conocimientos o habilidades a través de la experiencia.

[16] Pavlov fue Premio Nobel en 1.904 por su investigación sobre la digestión.

una psicología aplicada a problemas prácticos, que no se preocupa de los incomprensibles procesos mentales, sino que busca el *control y predicción* del comportamiento.

Hoy en día, es precisamente Watson el nombre más representativo de esta corriente, pero no fue, ni mucho menos, el único. Otro nombre ilustre es el de Edward L. Thorndike quien compartía sus ideas y enunció la *teoría del aprendizaje por ensayo y error* (en inglés, *drill and practice*). En ella, entra en juego la posibilidad de un *resultado negativo* a una determinada respuesta a un estímulo. Es decir, la relación estímulo-respuesta de Pavlov se enriquece al considerar el éxito o el fracaso de dicha respuesta. Ante un final satisfactorio el enlace se *refuerza*, y se debilita en caso contrario[17]. Aunque hoy *el ensayo y error* no siempre es considerado una buena técnica de aprendizaje, no puede negarse su importancia histórica[18]. Además, Thorndike también remarcó, por primera vez, la importancia de la *motivación*, y el concepto de *transferencia*, es decir la aplicación de un determinado conocimiento a nuevas situaciones.

Hoy en día, se dice que esta corriente de la psicología se basa en el *condicionamiento clásico*, cuya principal crítica es que se olvidaba por completo del raciocinio. De hecho, los experimentos que realizaban solían implicar animales: el propio Thorndike consideraba que visitar un aula era una pérdida de tiempo y que era mucho más científico experimentar con animales en su laboratorio que observar a los alumnos en clase.

Skinner (1938) supuso una bocanada de aire fresco a las ideas estancadas de Thorndike y Watson, originando una nueva ola en el conductismo. Durante los cincuenta años posteriores a su primera publicación, Skinner sometió sus teorías a elaboraciones, críticas y reelaboraciones sucesivas. Su mayor aportación al conductismo fue el que denominó *condicionamiento operante*, propio del hombre, y que *no* necesita estímulos, sino que actúa espontáneamente sobre el medio provocando en él consecuencias que determinan, a su vez, formas especiales de comportamiento. En este tipo de condicionamiento la pareja estímulo-respuesta desaparece, pues habla de acciones espontáneas al sujeto.

Dado que sus ideas siguen bajo el paraguas del conductismo, Skinner mantiene la opinión de que el condicionamiento operante, al igual que el clásico, puede ser *moldeado* mediante refuerzos. Sin embargo mejora las ideas de Thorndike al diferenciar nuevos tipos:

- *Refuerzo positivo*: cualquier consecuencia positiva que sirve para fortalecer una respuesta, como un halago o una buena nota.
- *Refuerzo negativo*: fortalecimiento de una conducta realizada por la

[17] El *final satisfactorio* no sólo incluye resultados materiales ante problemas *del medio ambiente*, sino también las palabras "bien" o "mal" de un profesor.

[18] Más adelante, veremos que en algunas cuestiones, Thorndike fue un visionario que se adelantó 50 años a su tiempo.

eliminación de consecuencias desagradables (por ejemplo, levantar un castigo).

- *Castigo I*: consecuencias *negativas* que intentan la *supresión* de una determinada conducta.

- *Castigo II*: desaparición de un hecho positivo que también intenta la *supresión* de una conducta (por ejemplo, quitar un premio, eliminar el recreo o la clase de gimnasia a un alumno).

Además, estudió los refuerzos no sólo en el ámbito de los dos tipos anteriores, sino también analizando lo que denominó *refuerzos secundarios* y *refuerzos parciales* para crear *programas de refuerzo* donde planeaba con cuidado los momentos en los que aplicar los premios con la intención de maximizar el *moldeamiento* de la conducta (Skinner, 1968).

Como se ha dicho, la corriente que Skinner comenzara a finales de la década de 1.930 se extendió durante buena parte del siglo XX, y tuvo gran repercusión en el sistema educativo, especialmente en el estadounidense. Como no podía ser de otra manera, no faltaron grandes detractores, lo que hace que a día de hoy sea difícil hacerse una idea de si realmente Skinner era un visionario o un loco (Rutherford, 2003).

2.4.2. Cognitivismo

El paso del tiempo y la recolección de nuevos resultados de investigación no beneficiaron al conductismo, que comenzó a no ser capaz de explicar ciertos aspectos del aprendizaje humano. A pesar de las mejoras de Skinner, la simplificación del comportamiento humano a la pareja estímulo-respuesta parece indicar que los sujetos nos limitamos a *responder de forma automática al ambiente*. Sin embargo, no siempre es necesario el largo proceso de ensayo y error, pues la *inteligencia* humana interviene decisivamente y, casi instantáneamente, puede superar una dificultad y encontrar la solución del problema.

En este sentido, los cognitivistas comienzan a dar importancia al *conocimiento*, y consideran que el *aprendizaje significativo* supone dar sentido a la información, organizarla, guardarla en la memoria y usarla para aprender otra nueva (Santituste Bermejo, 2003a). Así, el sujeto pasa de ser un mero autómata respondiendo estímulos para tomar un papel activo al *procesar la información*. El modo en el que esto tiene lugar sigue siendo inobservable, pero es posible plantearse los diferentes modos en el que el conocimiento se organiza en la mente para tratar de facilitar su adquisición. El papel del profesor se invierte, al dejar de tener el papel principal de poseedor de conocimientos y comenzar a ser un mero instrumento para fomentar e impulsar el aprendizaje. Aunque estas teorías llegaron después de su muerte, aquí es

aplicable la famosa frase de Albert Einstein *"Nunca enseño a mis alumnos; sencillamente intento conseguir las condiciones en las que puedan aprender"*.

Bajo este enfoque es donde toman sentido las diferentes técnicas de estudio como los mapas conceptuales, las clasificaciones, esquemas, etcétera, a las que suele conocerse como *estrategias metacognitivas*. Esto saca a la luz la necesidad de *enseñar a aprender* que hoy en día tan en boga está en los sistemas educativos. Sólo así se capacita al educando para que aprenda a *autoeducarse* a lo largo de toda su vida (Vázquez Gómez, 2003).

Aunque las teorías cognitivas han desarrollado multitud de modelos, hay dos que destacan por su importancia: el aprendizaje por descubrimiento, que debemos a Bruner (1966), y el constructivismo de Piaget (1955).

En el primero de ellos, el profesor no expone los contenidos de una manera acabada, sino que proporciona las herramientas básicas para que el individuo descubra por sí mismo lo que se desea que aprenda, incluso aplicando prueba y error. Su finalidad es atacar el aprendizaje memorístico, sustituyendo el aprendizaje de las fórmulas por el planteamiento de problemas.

Considerando que el descubrimiento consiste en transformar o reorganizar la experiencia de manera que se pueda ver más allá de ella, no cabe duda que este tipo aprendizaje sigue las ideas del cognitivismo. Suele ser un modelo bastante motivador que favorece la maduración del estudiante. Además, como el aprendizaje supone un cierto esfuerzo, una vez conseguido, el alumno lo valora mucho más, lo que supone que normalmente lo aprendido se transfiere de manera más natural a otras situaciones. Por desgracia, requiere mucho más tiempo, y exige *participación* por parte del alumno, que sólo se consigue si está lo bastante motivado.

Por su parte, el constructivismo postula que el saber existe en la mente como representación interna de una realidad externa. El aprendizaje es visto por lo tanto como la *construcción* de esa representación interna, de modo que el papel del profesor es ayudar a los alumnos a modelar sus propias perspectivas. Así, cualquier cosa que el alumno percibe se contrasta con su conocimiento anterior y, si encaja dentro del mundo que hay en su mente, puede formar nuevo conocimiento que se llevará consigo. Esto viene a indicar que el aprendizaje se trata más de un proceso de interpretación que de una transferencia de información de un cerebro a otro. Esta idea choca con la educación tradicional basada en el "modelo de la reproducción del conocimiento", dependiente de las clases verbales estructuradas y las sesiones de prueba y error donde el alumno es visto como un participante pasivo a la espera de ser "rellenado con conocimiento" (Papert, 1990). Dada la nueva visión del conocimiento como algo dinámico proveniente de múltiples fuentes, los estudiantes deben cambiar de papel y se convierten en actores activos de su propio aprendizaje. En este sentido, para conseguir un mejor aprendizaje no hay que buscar mejores formas de explicar, sino nuevas oportunidades para que el estudiante *construya* conocimiento.

De esta forma, el conocimiento se refuerza si el alumno puede usarlo con éxito en el entorno que le rodea, pues no es un mero banco de memoria que absorbe información pasivamente. En este sentido, se considera como la mejor forma de enseñanza a aquella que potencia el "aprender haciendo" (aprendizaje activo) siguiendo el viejo dicho de Confucio "escucho y olvido; veo y recuerdo; hago y comprendo". Por lo tanto, la enseñanza se basa en la resolución de problemas y ejercicios cada vez más complicados que potencien la construcción del conocimiento. Este tipo de aprendizaje puede realizarse de forma individual, o de forma social, mediante la interacción de varios estudiantes.

Dado que los estudiantes aprenden la nueva información que se les presenta *construyendo* sobre el conocimiento que poseen ya, es importante que los profesores se cercioren constantemente de que el resultado es el esperado. De hecho, los alumnos podrían sufrir lo que se conoce como *errores de la reconstrucción*, que son equivocaciones en el conocimiento creado cuando "llenan los agujeros" de su entendimiento con pensamientos que, aunque resultan lógicos, son incorrectos. Los profesores necesitan detectar e intentar corregir esos errores en la medida de lo posible. Curiosamente, cuando se encuentran, sólo *se facilita su resolución* (en lugar de solventarlos directamente), estimulando así la *auto-regulación* ya que son los propios alumnos quienes deben ser capaces de resolverlos por sí mismos. Por tanto, aunque las implicaciones pedagógicas tanto de conductistas como de cognitivistas exigen la comprobación continua por parte del profesor de que el aprendizaje está teniendo lugar, el conductismo busca averiguar si el alumno *sabe* (para utilizar refuerzo en caso contrario), mientras que el cognitivismo (y en concreto el constructivismo) intenta cerciorarse de que el alumno *entiende correctamente*.

2.5. El sistema educativo

Sea cual sea la teoría educativa y el método pedagógico utilizado, lo habitual es la enseñanza en clases a las que asisten un variable número de alumnos. Esto asegura que los estudiantes dedican una parte de su tiempo al aprendizaje, y además se potencia la convivencia entre ellos, lo que puede incitarlos a mejorar por el mero hecho de ser superiores a los demás (la conocida como *presión social*).

Sin embargo, la relación entre ellos puede llevar a la situación inversa. La influencia de peores estudiantes puede también crear *presión* sobre otros más aplicados, que tienen la desgracia de pertenecer a un grupo de amigos donde obtener buenos resultados está mal visto. En general, según Bennett (1999) la enseñanza en clases formadas por un grupo de alumnos tiene los siguiente problemas:

- El profesor debe adaptar su velocidad y el nivel de la enseñanza en función a la media de la clase. Eso hace que los estudiantes deban adaptarse a planes de estudios fijos, ligeramente moldeados por el profesor para adecuarlos a las características de otros 20 ó 30 compañeros. Como consecuencia, los alumnos aventajados aprenderán deprisa, y desaprovecharán mucho tiempo. Por su parte, los alumnos más lentos serán incapaces de conseguir entender todo correctamente, entrando en un proceso acumulativo que tiene consecuencias desastrosas.

- Es imposible conseguir que todos los alumnos obtengan los mismos conocimientos.

- Proporcionar un aprendizaje motivante requiere una adaptación de los contenidos a los intereses, capacidades, y facilidad de aprendizaje de cada alumno. Dicha adaptación no puede realizarse en clases compartidas, lo que ocasiona que muchos estudiantes pierdan la motivación y encuentren las clases monótonas y carentes de interés.

- Aunque teóricamente el profesor permite que cualquier alumno pregunte sus dudas, la realidad es que habitualmente sólo los más adelantados lo hacen, al estar seguros de que ninguno de sus compañeros conoce la respuesta. El resto, temen hacer una pregunta demasiado sencilla que les deje en evidencia. Alumnos aún con menos conocimientos pueden, incluso, ser incapaces de saber qué preguntar.

Debido a que, finalmente, el método de enseñanza del profesor se adapta a la media de la clase, es sencillo vislumbrar dos grandes grupos perjudicados por la enseñanza en escuelas: los alumnos aventajados y los más lentos en el aprendizaje. Los primeros tienen el problema potencial del aburrimiento y la pérdida de tiempo. Al comprender los conceptos rápidamente, se ven obligados a esperar a que el resto de la clase los asimile antes de continuar con nuevos retos para su inteligencia. Es universalmente conocido que los niños superdotados suelen tener problemas de comportamiento en clase debido precisamente a esto.

Por su parte, los alumnos más lentos pueden llegar a ser incapaces de comprender a tiempo lo que se les está enseñando. Eso hace que les sea completamente imposible asimilar los conceptos más complicados que vendrán después, entrando en una espiral negativa de la que puede resultar muy difícil salir (Skinner, 1958). La aparición de estudios que aseguran que repetir curso daña la autoestima de los niños ha supuesto la implantación de ciertos sistemas educativos que permiten avanzar a pesar de no haber superado los niveles previos. Eso hace que, salvo en algunas ocasiones, los nuevos contenidos tampoco sean asimilados, lo que finalmente puede crear en estos alumnos una gran frustración que los lleva a sentirse incapaces de terminar

su formación. Lo realmente lamentable es que esa situación podría haberse solucionado si se hubieran tomado medidas a tiempo.

La solución que algunos plantean es separar a los alumnos en diferentes grupos en función de sus características. De ese modo, el nivel de cada clase está más equilibrado, y los profesores pueden adaptar la velocidad y el contenido al interés de un mayor número de alumnos. Idealmente, todos salen beneficiados: los alumnos reciben una educación más adecuada, y los profesores no deben tratar simultáneamente en la misma clase con estudiantes con muy diferentes cualidades.

Sin embargo esta solución hay que verla con cuidado. Al hacer estos grupos, se establece de forma explícita la diferencia entre los alumnos "listos" y los alumnos "tontos", lo que resulta perjudicial para la relación entre ellos, y para la autoestima de los menos afortunados. Desde el punto de vista de los profesores, la división puede generar prejuicios inconscientes que reducen su motivación cuando deben enseñar a los grupos inferiores. Eso ocasiona que la enseñanza que éstos reciben sea de peor calidad y con un nivel de exigencia inferior al que debería, por culpa de una mal entendida compasión. Todo esto origina que los alumnos catalogados en los grupos menos favorecidos tengan muchas dificultades para salir de ellos.

A la vista de todo esto, resulta obvio que la solución son las clases individuales. Un profesor particular tiene las siguientes ventajas:

- Puede adaptar el curso a los intereses y conocimientos del alumno.
- Es consciente de lo que el estudiante sabe y lo que no, lo que le permite evitar repetición.
- Si el alumno enferma, el curso continuará exactamente en el punto donde se quedó.
- Idealmente, el profesor animaría al alumno continuamente, remarcando sus éxitos y quitando importancia a los fracasos. Además, no debería perder la paciencia en el caso de tener que repetir una y otra vez la misma explicación si el alumno es incapaz de comprenderla.

De todas las anteriores, la ventaja principal es la primera. Debido al trato individualizado, un buen profesor puede ser capaz de motivar al alumno y mantenerle interesado durante todo su aprendizaje. A la vista de todo esto, no resultan extraños los datos revelados por Bloom en 1984, quien encontró que los estudiantes que reciben una enseñanza individualizada consiguen unos resultados con una desviación típica dos veces superior a los de aquellos que son educados en grupo.

2.6. Enseñanza asistida por ordenador

Por desgracia, el establecimiento de las clases particulares como modelo de enseñanza no es viable, pues es excesivamente caro. Sin embargo la llegada de los ordenadores a las escuelas ha ocasionado que muchos se pregunten si éstos pueden sustituir a los profesores para proporcionar una enseñanza individualizada. En este sentido, se denomina *enseñanza asistida por ordenador* a la educación que utiliza a los ordenadores como un medio (Lowe, 2001), y cuya principal ventaja es proporcionar un modo alternativo para que los estudiantes consigan sus objetivos de forma individual, con una experiencia de aprendizaje autodirigida y a su propia velocidad.

Hoy en día, la enseñanza asistida por ordenador incluye los sistemas de enseñanza, el material multimedia, e incluso el uso de la red para apoyar el aprendizaje. Esto hace que el concepto agrupe a un amplio rango de aplicaciones que, a lo largo de los años, han ido recibiendo un nutrido número de acrónimos con dudosas diferencias (Rist y Hewer, 1996):

- CAI: *Computer Aided Instruction*, instrucción asistida por ordenador.
- CAL: *Computer Aided Learning*, aprendizaje asistido por ordenador.
- CBL: *Computer Based Learning*, aprendizaje basado en ordenador.
- CBT: *Computer Based Teaching* o *Training*, enseñanza (o entrenamiento) basada en ordenador.
- CAA: *Computer Aided Assessment*, evaluación asistida por ordenador.
- CMC: *Computer Mediated Communications*, comunicación a través de ordenador.

Independientemente de la sobrecargada terminología, los más visionarios pueden plantearse la posibilidad del uso de ordenadores para conseguir una total sustitución de los profesores. En este sentido y, de nuevo, según Bennett (1999), un programa con alumnos conflictivos realizado en Florida en 1.987 demostró que se pueden utilizar ordenadores para sustituir completamente al tutor. Si el sistema es lo suficiente bueno, podría tener las mismas ventajas pedagógicas que un profesor particular. Además los ordenadores disponen "por naturaleza" de una paciencia infinita y una falta total de prejuicios, lo que les hace especialmente adecuados para enseñar a alumnos lentos. A esto, se añade la creencia habitual indicada por Mark y Greer (1993) de que los ordenadores son "intrínsecamente motivantes" o que, según Emihovich y Miller (1988), mejoran la autoestima. Por último, generalmente un alumno se siente menos intimidado por un ordenador que por un profesor particular que le está evaluando continuamente. Por supuesto el ordenador también

podría estar permanentemente valorando cada acción del estudiante, pero es más difícil que éste sea consciente de ello.

El experimento de Florida consiguió unos resultados sorprendentes. Similares experiencias posteriores parecen indicar que, de media, el aprovechamiento por parte de los alumnos de 2 horas de enseñanza por ordenador es similar al de un mes de clase. No obstante, hay que considerar los resultados con cuidado. Los alumnos a los que se les aplicó el programa eran aquellos que, estando en cursos superiores, sufrían un acentuado retraso. Lo que se hizo fue darles la oportunidad de aprender lo que se habían perdido y continuar a partir de ahí. Estas cifras son increíblemente altas, y no pueden conseguirse con estudiantes menos problemáticos. Sin embargo, según Ong y Ramachandran (2000) los resultados obtenidos con alumnos "convencionales" también son prometedores.

Naturalmente, al menos hoy, construir sistemas que puedan sustituir completamente al profesor es imposible. Un ordenador no puede hacer ciertas valoraciones que un profesor sí puede, como por ejemplo evaluar la calidad de una redacción. Además, los profesores no sólo enseñan cultura, también son educadores (dedican esfuerzo al resto de *dimensiones* de la educación, tal y como ya hemos dicho). Por tanto, en la práctica el uso de ordenadores como tutores *no elimina* a los profesores; sencillamente cambia su papel. De hecho, en el experimento de Florida, eran los ordenadores quienes enseñaban, pero los alumnos *siempre* tenían al alcance a profesores que les apoyaban. La enseñanza seguía realizándose en centros escolares para aprovechar sus ventajas, y los profesores tenían como papel principal proporcionar una relación más humana dentro del proceso, sustituyendo al ordenador en aquellas tareas que éste era incapaz de llevar a cabo. En resumen, los ordenadores se utilizaron para la *dimensión intelectual* de la educación. Pero seguía utilizándose el mismo entorno educativo para el resto de dimensiones.

Al fin y al cabo, no hay que olvidar, que si bien la educación individualizada es positiva para el aprendizaje del *material*, la falta de contacto con los semejantes impide el *aprendizaje social* que nos permite desenvolvernos en la sociedad, saber cómo tratar a los demás, y sentirnos parte de un grupo (de hecho, una de las necesidades primarias de Maslow). Una posible solución es la dada por Schery y O'Connor (1997) en un estudio con niños con el que argumentan que si los alumnos utilizan estos sistemas por parejas se consigue el desarrollo de habilidades sociales y el mantenimiento de conversaciones al practicar el cambio de papel de oyente a hablante y la capacidad de escuchar. A pesar de esta opinión, es habitual la creencia expuesta por Lowe (2001) sobre que la educación asistida por ordenador afecta de manera positiva en los resultados de los estudiantes cuando se compara con la enseñanza clásica en clase, pero sólo debe utilizarse como un apoyo a ésta, no como un reemplazo.

En realidad, hemos estado asumiendo continuamente el uso de los orde-

nadores como profesores cuando, según Taylor (1980), éstos pueden utilizarse de tres formas para apoyar la educación:

- *Como tutor*: el ordenador hace las veces de profesor al mostrar el contenido y guiar al alumno en el aprendizaje.
- *Como herramienta*: los alumnos utilizan el ordenador para simplificar su labor o mejorarla. En esta categoría entra, por ejemplo, el uso de los procesadores de texto o las hojas de cálculo. Desde el punto de vista del profesor, también se ha extendido el uso del ordenador especialmente en la presentación de transparencias o demostraciones interactivas (Gómez Martín y Gómez Martín, 2006).
- *Como tutelado*: es conocido el efecto que sobre una persona tiene la obligación de *explicar* algo: normalmente el esfuerzo mental necesario (y quizá la preocupación a preguntas no previstas) hace que lo explicado se comprenda mejor. Esta idea puede usarse si se construye un sistema con el que el ordenador hace las veces de *alumno*, y el estudiante hace las veces de *profesor* "enseñando" algo al ordenador.

El último grupo es una buena idea, pero su gran dificultad de implementación supone que no haya demasiado trabajo realizado sobre él. Centrándonos en la primera de las tres clasificaciones, tradicionalmente se considera que existen tres tipos de sistemas en los que el ordenador hace las veces de profesor (Anderson, 1986):

- *De repetición* (del inglés *Drill and practice*): se basan en la repetición estructurada de conceptos *aprendidos previamente*. Utilizan interacciones del tipo pregunta-respuesta, y confían en que la repetición continuada refuerce el aprendizaje. Ejemplos típicos son aplicaciones para que los niños refresquen las tablas de multiplicar, para repasar vocabulario de idiomas extranjeros, o para practicar tests del examen de conducir.
- *Tutores*: mejoran los anteriores porque se utilizan para aprender *conceptos y procesos nuevos*, no sólo para reforzarlos. El sistema presenta el material de una manera estructurada y suele permitir ponerlo en práctica para evaluar su comprensión.
- *Simuladores*: modelizan una situación real o imaginaria, y permiten experimentar con ella. Suelen basarse en gráficos interactivos, y permiten que el alumno explore de una manera segura los efectos del cambio de parámetros en el funcionamiento del sistema que se simula.

En realidad, incluir los *simuladores* dentro del grupo de uso de ordenadores como *profesores* es conflictiva. Podríamos también considerar que los

simuladores son meras *herramientas* de apoyo al estudio, aunque quizá sea un poco excesivo encasillarlas en el mismo grupo que un procesador de texto. Debido a ello Seifert (1986) crea para ellos una clasificación independiente, separada de las herramientas y del resto de aplicaciones como tutores.

En lo que se refiere a las herramientas de apoyo, típicamente se piensa en lo que Rist y Hewer (1996) denomina "herramientas de producción", como los mencionados procesadores de texto, hojas de cálculo, bases de datos, o paquetes de presentación y publicación. Son herramientas generales, pero que aplicadas a entornos educativos pueden ser beneficiosas. Por ejemplo, un procesador de texto facilita la creación de borradores, lo que a su vez potencia la reflexión. Hoy también están en este grupo las herramientas de comunicación (el correo electrónico, o los entornos para intercambiar información como foros o salas de chat), y también los sistemas de recuperación de información (como los diccionarios o enciclopedias electrónicas).

En lo que a nosotros respecta, lo que más nos interesa aquí es el uso de ordenadores como *profesores*, es decir la primera de las tres clasificaciones de Taylor. Además, salvo para casos muy concretos como en la educación a distancia, hay que volver a recordar que suelen utilizarse *como apoyo* a la enseñanza habitual, no como sustituto. Esto tiene la ventaja adicional de que es mucho más fácil conseguir que el sistema educativo vaya permitiendo, poco a poco, la entrada de este tipo de sistemas. Generalmente el campo de la enseñanza es receloso a grandes cambios, debido a su alto riesgo: si el resultado obtenido no es bueno, el remedio puede ser peor que la enfermedad[19].

Según Lowe (2001), al igual que la Segunda Guerra Mundial fue un impulso para el desarrollo de los ordenadores, la guerra de Vietnam lo fue para el uso de ordenadores como tutores. La necesidad de entrenamiento militar llevó al ejército estadounidense a solucionar la carencia de instructores con sus "grandes" ordenadores. El resto del capítulo se dedica a recorrer los principales tipos de sistemas que tradicionalmente se han desarrollado para convertir a los ordenadores en tutores, y veremos que los primeros esfuerzos surgieron en realidad mucho antes de la guerra de Vietnam, aunque es cierto que, según Molnar (1997), el primer proyecto a gran escala se realizó en 1.959 en la Universidad de Illinois (la guerra de Vietnam comenzó en 1.958 y terminó en 1.975). Esos primeros sistemas, sigue Lowe, estaban influenciados por las teorías conductistas del aprendizaje, pero cuando la tecnología subyacente mejoró, se produjo un claro desplazamiento hacia las cognitivistas.

[19] Esta opinión choca, sin embargo, con la triste realidad de las sucesivas reformas educativas que nuestro país ha venido soportando (LOGSE, LOCE y más recientemente LOE).

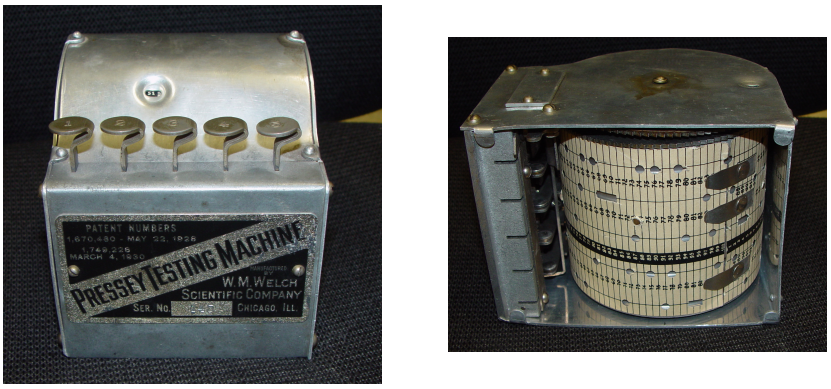


Figura 2.1: Máquina de tests de Pressey (extraídas de Wikipedia)

2.7. Sistemas de enseñanza basados en marcos

2.7.1. Antecedentes históricos y definición

Las ideas de Thorndike (descritas en la página 21) sobre el uso de *prueba y error* en el aprendizaje empujaban hacia técnicas pedagógicas en las que los alumnos debían realizar gran cantidad de ejercicios y pruebas hasta que fueran capaces de responder correctamente a las preguntas enunciadas por el profesor. Sobre éste recaía la pesada tarea de repetir y corregir continuamente a sus alumnos hasta que el reiterado *refuerzo* de la respuesta correcta a cada pregunta (*estímulo*) les llevara a responder siempre de manera acertada.

Para aliviar esa carga, Pressey fabricó en 1926 un dispositivo mecánico para plantear preguntas tipo test a los alumnos y, más importante aún, para *corregir* sus respuestas de manera automática (figura 2.1). Funcionaba con papel perforado para indicar las respuestas, y con un contador que se iba incrementando por cada acierto. Hoy clasificaríamos a este dispositivo sin ningún género de dudas como un sistema de enseñanza por repetición (*drill and practice*).

A lo largo de los años (1927 y 1932), Pressey modificó su máquina para poner en práctica las teorías del refuerzo de Thorndike, añadiendo diversas variaciones como la posibilidad de que la máquina *no* pasara a la siguiente pregunta hasta que la anterior no hubiera sido acertada. Así, el dispositivo pasó de ser promocionado como una máquina para hacer test a serlo como una *máquina para enseñar*. Pressey soñó con una "revolución industrial" en la educación gracias a su idea, que no llegó a ser tal por la llegada de la Gran Depresión.

Aunque fuera Pressey el primero que consiguió fabricar una máquina de apoyo a la enseñanza, realmente fue el propio Thorndike (1912) quién había tenido la visión de los primeros sistemas automáticos de enseñanza. Las páginas 164-165 de su libro son premonitorias de lo que tardaría aún 50 años

en llegar. Criticaba los libros tradicionales por limitarse a proporcionar el resultado de los razonamientos de los autores y a plantear nuevos ejercicios, pero no ser capaces de llegar más lejos y *analizar* si el alumno había o no aprendido ya lo suficiente, o proporcionar ayudas y guías en función de las trabas que cada lector pudiera encontrarse.

Obviamente en aquella época, en la que el papel impreso era la forma más asequible de comunicación y cualquier otro avance tecnológico no había alcanzado aún el horizonte, tenía que conformarse con culpar a los lectores de su incapacidad para *seguir las instrucciones* que un manual pudiera darles. El mayor problema que veía era la clara posibilidad que había de que un alumno *engañara* a un libro de ese tipo. Soñaba con la existencia de libros que, gracias a alguna maravilla de la técnica, pudieran *impedir* que una página posterior fuera vista por alguien que no hubiera demostrado haber leído y comprendido la anterior[20]. En realidad, el problema de fondo está en la *pasividad* de un lector frente a un libro, que no puede interactuar con él de ninguna forma. Según The Economist (2005) el propio Sócrates criticó esto aunque de una manera mucho más radical. Él vivió en un mundo en el que los libros no estaban aún integrados en la sociedad, por lo que debió existir un amplio debate sobre su utilidad. La opinión de Sócrates (publicadas por Platón en *Phaedrus*) eran drásticas. Opinaba que la sustitución de la tradición oral por los libros crearía una irremediable pérdida en el alma de los aprendices, al dejar de utilizar la memoria y pasar a confiar en las letras escritas de los libros en lugar de recordarlas por sí mismos. Además, la versión escrita de un diálogo no podría sustituir a la habilidad de interrogar al oyente, pues la reacción siempre sería la misma, independientemente de la respuesta.

En cualquier caso, y volviendo al siglo XX, el dispositivo de Pressey falló al carecer de la habilidad de *mostrar contenido*, dado que se limitaba a realizar y comprobar preguntas. De ahí la crítica que Skinner (1958) hiciera al indicar que dicha máquina necesitaba que el aprendizaje se hubiera realizado de alguna manera diferente al propio uso de la máquina. Esta opinión deja ver entre líneas las dudas que Skinner tenía sobre el condicionamiento clásico, tal y como se ha comentado ya.

A pesar de esta crítica, Skinner apoyaba incondicionalmente este tipo de iniciativas. De hecho, excusó el escaso éxito de Pressey por haber llegado demasiado pronto, en un mundo con demasiada "inercia" en la psicología del aprendizaje donde sólo se potenciaba la memoria. Pero no dudaba en su importancia al remarcar por primera vez la importancia de la *realimentación inmediata*, y la posibilidad de que cada alumno avanzara a su propia velocidad.

[20]Sus palabras textuales (Thorndike, 1912, página 165) fueron "If, by a miracle of mechanical ingenuity, a book could be so arranged that only to him who had done what was directed on page one would page two become visible, and so on, much that now requires personal instruction could be managed by print."

De hecho, el propio Skinner amplió el trabajo de Pressey en lo que denominó *enseñanza programada*. Al parecer, decidió aplicar sus teorías del condicionamiento operante a la educación tras asistir, el día del Padre, a una clase de aritmética de su hija donde fue testigo de "cómo se destruían las mentes de los alumnos" (Smith, 1994; Skinner, 1983). Creyó que los alumnos podrían ser condicionados con el aprendizaje paso a paso reforzando cada respuesta correcta hasta la adquisición de formas de conductas más complejas (Santituste Bermejo, 2003b).

Para eso, proponía el uso de *máquinas de enseñanza* para solventar la mala *escalabilidad* que tenía el sistema educativo. Con el aumento de la población y de su inquietud por el conocimiento, la enseñanza cada vez era más demandada; para abastecer esa necesidad, la *fuerza bruta* de crear más escuelas y formar más profesores debía ser revisada para *optimizarla* tanto en recursos como en calidad. La enseñanza *individualizada* tenía que imponerse, y para eso se necesitaban medios tecnológicos. En aquella época (estamos hablando de 1958), comenzaban a usarse tímidamente los medios audiovisuales, pero no proporcionaban una adaptación a la velocidad de cada alumno que, además, seguían teniendo un papel completamente *pasivo*.

La idea de Skinner era similar al sueño de Thorndike. Animaba a la construcción de máquinas que hicieran a los alumnos avanzar por una secuencia de pasos diseñada cuidadosamente, cada uno de ellos de una longitud tal que pudiera ser realizada sin interrupción, pero que, en conjunto, llevaran al estudiante hacia la adquisición del comportamiento deseado.

Por lo tanto, en conjunto, la idea de la enseñanza programada no es más que un conjunto de material organizado en porciones o "marcos" (*frames*), cada uno de ellos presentado por separado. El paso de uno a otro es decidido de manera automática en función de si el alumno demuestra o no que ha comprendido el anterior. Se realizaron algunas máquinas mecánicas que lo soportaron. Naturalmente, Skinner no se interesó directamente por la tecnología de estos dispositivos, sino sobre *cómo usarlos*, es decir, cómo preparar el *material* que se incluía en ellas y que era el que, realmente, permitía enseñar. Sus ideas, al igual que su condicionamiento operante, se sustentaban en la repetición, la actividad y en la progresión continua de los alumnos, que recibían enseñanza individualizada.

Sus aportaciones fueron bastante criticadas en aquella época, generando incluso sátiras en revistas de humor (Smith, 1994). No obstante, las ideas de la enseñanza programada, que pueden aplicarse manualmente sin el uso de dispositivos, se muestran especialmente útiles en la enseñanza de matemáticas y de idiomas. Según Vargas y Vargas (1991), antes de su divulgación en la década de 1.960, la mayor parte de las aproximaciones pedagógicas se limitaban a la transmisión de información, asumiendo que el trabajo de la enseñanza terminaba una vez que el material era correctamente organizado y presentado. Si un determinado estudiante *no* aprendía, él era el único

culpable, porque la labor del profesor había sido intachable.

Skinner consiguió un cambio de mentalidad en los docentes de todo el mundo. Hoy en día su *enseñanza programada* está implícita en gran cantidad de sistemas informáticos educativos a los que Freedman et al. (2000) denomina *sistemas de enseñanza basados en marcos* (frames). En realidad este nombre es utilizado únicamente por algunos autores. Dado que históricamente fueron los primeros en llegar, fue con ellos con los que se acuñaron los primeros acrónimos, como CAI (*Computer Assisted Instruction*) o CBT (*Computer Based Teaching*). Debido a ello, estas siglas siguen utilizándose hoy para referirse a ellos, si bien su significado es mucho más general al haberse ampliado el espectro de alternativas. Otro nombre que se ha hecho común es *e-learning*, aunque habitualmente este término lleva implícita la idea de que se realiza mediante Web, y se enriquece con las nuevas alternativas que ésta proporciona.

Como ya se ha esbozado previamente, el funcionamiento más básico de este tipo de sistemas se basa en la división del contenido en unidades pequeñas (marcos o *frames*) con un objetivo de aprendizaje concreto (McKenna y Laycock, 2004). Cada uno de ellos se muestra de manera independiente, y requiere una sencilla intervención del alumno para demostrar que lo ha comprendido. Sólo cuando responde correctamente se le permite pasar al siguiente marco. Aunque en este modelo el contenido es completamente lineal, al menos estos tipos de sistemas pueden considerarse *tutores* (aunque sencillos), pues son capaces de *mostrar contenido* (es decir, dejan de ser meros sistemas de repetición o *drill and practice*).

Un claro punto de mejora sobre este modelo básico es permitir *ramificaciones* para adaptar el contenido del curso en función de las necesidades del alumno, intuidas a partir de sus respuestas. Es decir, en cada marco, el alumno se enfrenta a una prueba para comprobar que ha entendido el contenido. Si da la respuesta correcta, se avanza. Si no, en lugar de forzar al estudiante a volver a leer el mismo contenido y repetir el mismo ejercicio, es posible redirigirle hacia un marco nuevo, donde se le proporcione información *contextualizada* dependiente del error concreto.

Estos sistemas tienen por lo tanto el conocimiento "cableado" en los enlaces. Así, el programa es capaz de mostrar cada una de las partes en las que se ha dividido el material, y su toma de decisiones está implementada con pares condición-acción del estilo de "si se responde correctamente a la pregunta 12, continuar con el material y pregunta 24; en caso contrario, saltar a la 14" (Beck et al., 1996; Freedman et al., 2000).

Este tipo de sistemas permiten a los alumnos un aprendizaje más rápido al proporcionar una respuesta inmediata a sus intervenciones (Sloane et al., 1989). Además, tal y como dicen McKenna y Laycock (2004), hacen uso de interfaces sencillos que proporcionan una interacción simple. Debido a eso,

son mucho más rápidos de crear, lo que los convierte en el tipo de sistemas educativos más numerosos. La usabilidad, predicibilidad, claridad estructural y facilidad para indicar la "posición actual" (debido al uso de contenido claramente lineal) facilita su aceptación entre los usuarios (especialmente entre aquellos con poca experiencia en el uso de ordenadores). Esto es ventajoso porque alumnos de corta edad pueden utilizar este tipo de sistemas como primera introducción a la informática. Así, los pueden utilizar para aprender conceptos propios de su edad (por ejemplo, los colores) a la vez que se familiarizan con los ordenadores. Además, es beneficioso para los *propios profesores*. Según Labbo et al. (2003), en Estados Unidos disponían de media en 2.002 de un ordenador con conexión a Internet por cada 5'6 alumnos. Sin embargo, al menos el 50 % de los profesores confesaban sentir dificultades para utilizarlos, por lo que estos interfaces sencillos son de agradecer.

No obstante, y de manera más bien anecdótica, aunque este tipo de sistemas basados en marcos se implanten hoy haciendo uso de ordenadores, el funcionamiento es tan simple que, de hecho, podría implantarse en un libro. En realidad el uso de ordenadores consigue evitar la posibilidad de que los alumnos mientan al sistema adelantándose a partes del contenido más allá de lo que han demostrado comprender, es decir, precisamente lo que Thorndike soñó en la segunda década del siglo XX. No obstante, hoy su funcionamiento a muchos nos recuerda a la serie de libros "Elige tu propia aventura" (*"Choose your own adventure"* en inglés) cuyo primer libro fue publicado en 1.979, y que crean historias a las que Aldrich (2005) denomina "historias ramificadas" (*branching stories*).

2.7.2. El hipertexto y la hipermedia

Esta idea de dividir el contenido en diferentes unidades independientes hoy en día nos recuerda también al *hipertexto*. Éste consiste en texto escrito de manera *no secuencial*, que permite saltos y recorridos por diferentes caminos seleccionables por el usuario. El padre de esta idea, aplicada a la *gestión de conocimiento*, fue Bush quien, en 1945, imaginó un dispositivo, al que bautizó Memex, con forma de mesa de despacho donde mantener ingentes cantidades de documentos microfilmados y *relacionados* entre sí para una mejor búsqueda de información (figura 2.2). Incluso proponía la aparición de una nueva profesión a la que denominó *trail blazers* ("abridores de caminos") que se dedicaría a recorrer y relacionar los diferentes documentos. Todo esto permitiría una búsqueda mucho más eficiente, y liberaría al hombre de los índices alfabéticos que sólo permiten un "eje" de búsqueda cuando la mente humana, dijo, trabaja por asociaciones.

Según Lamarca Lapuente (2006), esta idea resultó revolucionaria al exponer por primera vez la posibilidad de un texto *multisecuencial* (recuérdese que Thorndike siempre habló de contenido completamente *lineal*, pero de

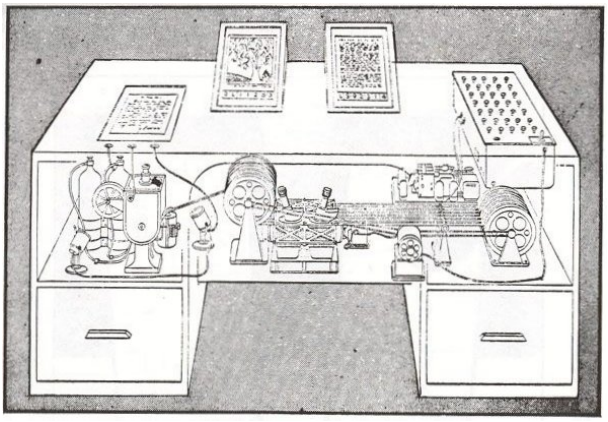


Figura 2.2: Idea original de la máquina Memex de Vannevar Bush

avance controlado). No obstante, fue Nelson (1965) quien acuñó el término *hipertexto* y lo definió como:

> "Por hipertexto entiendo escritura no secuencial. La escritura tradicional es secuencial por dos razones. Primero, se deriva del discurso hablado, que es secuencial, y segundo, porque los libros están escritos para leerse de forma secuencial... sin embargo, las estructuras de las ideas no son secuenciales. Están interrelacionadas en múltiples direcciones. Y cuando escribimos siempre tratamos de relacionar cosas de forma no secuencial."

En realidad, aquí ya la idea original del uso de hipertexto para la *gestión de conocimiento* y la ayuda a la búsqueda de documentos se ha degenerado en cierto modo, pues se orienta más bien a la escritura de *hipertexto* desde el principio, en lugar de a *abrir caminos* relacionando documentos previamente existentes. La palabra escrita había sido secuencial durante 3.000 años, y de repente el mundo fue consciente de que no tenía por qué ser así. Un hipertexto bien escrito anima a lo que hoy se conoce como *navegar* en lugar de a leer de principio a final (Woolf, 1992a), pero siempre manteniendo un cierto cuidado para evitar el efecto que Conklin (1987) denominó *lost in hyperspace* ("pérdida en el hiperespacio").

La idea del hipertexto fue muchos años después exitosamente reutilizada en el CERN[21] por los creadores de la World Wide Web (Cailliau y Gillies, 2000). Curiosamente, este uso del hipertexto se aleja de la idea original de la máquina Memex en la que los propios *usuarios y lectores* podían *abrir nuevos*

[21] El CERN (`http://www.cern.ch`) es el laboratorio físico de partículas más grande del mundo

caminos creando enlaces y completando información. La llegada de las páginas Web creadas de manera colaborativa (cuyo estandarte es la Wikipedia[22]) recuperó ese concepto inicial.

Una vez recordado el significado del término hipertexto, vemos que efectivamente los sistemas de enseñanza basados en marcos con enlaces *cableados* tienen un gran parecido. De hecho, Freedman et al. (2000) reconoce que estos sistemas son en realidad hipertexto con un objetivo educativo.

Un avance tecnológico que mejora tanto el hipertexto como los primeros CAI es la inclusión de multimedia, es decir de información diferente al texto. Esto desemboca en lo que se conoce como *hipermedia*. Permite mezclar texto con imágenes, sonido y videos, enriqueciendo el contenido.

Quizá la primera aparición de la multimedia llegó en 1.986 con el CD-I, el formato CD Interactivo desarrollado por Philips y Sony. El estándar se especificó en lo que se llamó *el libro verde*, y consistía en un uso específico del soporte Compact Disk (CD), donde se almacenaba audio, vídeo y texto en diferentes formatos. Para ser reproducido requería la compra de dispositivos especiales que se unían a la televisión. La gran ventaja de este sistema es que permitía que el usuario *interactuara* con él, indicando qué quería hacer en cada momento al estilo del hipertexto o los sistemas CAI. Estaba claramente enfocado al entretenimiento y a la educación. No obstante, la llegada masiva de las consolas de videojuegos hizo que la parte lúdica del sistema quedara en clara desventaja, por lo que su uso nunca llegó a despuntar.

El avance del *hardware* permitió la entrada en los hogares y colegios de ordenadores con capacidades de audio y gráficos inimaginables unos años atrás. Por desgracia, tanto audio como vídeo ocupan mucho espacio al ser digitalizado, por lo que hubo que esperar a la llegada de un sistema de almacenamiento secundario masivo, papel que cumplió a la perfección el CD-ROM. Con esta mezcla, el mercado se llenó de supuestos programas educativos interactivos y multimedia, que unían, en cierto modo, las ideas del hipertexto con una gran cantidad de espacio de almacenamiento. Ante esa abundancia de capacidad, varios ordenes de magnitud mayor que los dispositivos de almacenamiento anteriores, numerosas empresas se lanzaron a editar programas para enseñar ofimática, o enciclopedias interactivas ya fueran generales o específicas sobre temas concretos, como historia, inventos, el cuerpo humano, historia del arte, geografía, y un largo etcétera. El gancho para la venta (de forma aislada, o como "regalo" en revistas) era el número de vídeos, de documentos gráficos y sonoros y, en definitiva, de la abundante información disponible. Finalmente esa increíble cantidad de datos la mayoría de las veces resultaba ser de poca utilidad, pero la novedad hizo que mucha gente decidiera comprar o colec-

[22]La wikipedia es una enciclopedia libre plurilingüe basada en la tecnología Wiki. Es escrita de manera colaborativa por voluntarios permitiendo que la gran mayoría de las páginas sean modificadas por cualquier usuario utilizando un navegador Web. Está disponible en `http://www.wikipedia.org/`

cionar este tipo de programas que la mayoría de las veces no se llegaban a utilizar.

Y es que estos sistemas la mayoría de las veces no son más que libros electrónicos que explotan las capacidades multimedia, y que para sacarles provecho deben ser leídos (escuchados o vistos) casi como cualquier otro libro. Por lo tanto suelen recibir el mismo uso que los libros normales (bastante poco), convirtiéndose así en lo que a menudo se conoce como *shelfware*[23]. Estos sistemas, además, *no* suelen disponer ni siquiera de las técnicas de los primeros CAI en los que el contenido se organiza y se anima a su recorrido controlado. El resultado son programas educativos que difícilmente educan. Y es que al igual que nadie espera que si se deja a un grupo de alumnos en una biblioteca, tras un tiempo esos alumnos hayan adquirido una cierta educación (Bennett, 1999), tampoco puede esperarse que se eduquen viendo vídeos y escuchando narraciones sin una estructura o alguien que les guíe.

Esta visión un tanto crítica de la multimedia no significa que no sea útil; sencillamente los primeros usos no fueron satisfactorios, ya que los sistemas se acercaban más al hipertexto mal escrito que a los CAI con contenido bien estructurado. Hoy se utiliza con más cuidado y proporciona unos resultados más satisfactorios.

2.7.3. Herramientas de creación. La Web

Lo anterior demuestra que realmente lo que diferencia un sistema de otro suele ser *el contenido*. Esto marca una clara diferencia entre el sistema que da soporte a la educación, y el propio material. Si nos remitimos a la máquina original de Pressey, esta separación también existía, entre el dispositivo físico y los propios tests codificados en papel perforado.

Con el tiempo, han ido surgiendo diferentes herramientas para la generación de contenido educativo orientado a sistemas basados en marcos. Esas herramientas no tienen por qué ser específicas para ellos. Por ejemplo, aunque sea de una manera degenerada, cualquier programa para generar presentaciones (como el prematuro Harvard Graphics, o los posteriores Power Point y Open Office Impress) pueden utilizarse para generar contenido educativo interactivo si cada transparencia incluye un pequeño test para determinar si el usuario ha comprendido lo que se le ha expuesto.

El primer sistema del que se tiene noticia para crear contenido educativo es PLATO (*Programmed Logic for Automated Teaching Operations*), cuya primera versión data de finales de la década de 1.950. Su tercera iteración, de finales de los 60, permitía diseñar nuevas lecciones con el lenguaje de programación bautizado como TUTOR. Muchos otros sistemas específicos han

[23]Este término se utiliza para referirse al software que no se consigue vender, o el que se compra pero que no resulta útil o no se le da uso. En cualquiera de los dos casos, el resultado es un paquete de software que se mantiene en una estantería sin ser utilizado, incluso sin ni siquiera llegar a ser abierto

surgido con los años, hasta llegar a los actuales Macromedia AuthorWare[24], Tool Book[25] o MALTED (Multimedia Authoring for Language Tutors and Educational Development[26]) con los que se consigue una riqueza multimedia muy llamativa.

En la década de 1.990 se produjo una gran conmoción en este área con la llegada masiva del hipertexto y la hipermedia a los hogares de la mano de la Web. Como se ha comentado ya, estos sistemas son en realidad hipertexto con objetivos pedagógicos. La explosión de la Web hizo que el mercado se replanteara su modo de funcionar. Por primera vez, existía un estándar *de facto* para representar el contenido (estamos hablando, naturalmente, de HTML). Y, lo que es mucho más importante, surgió la posibilidad de que el material fuera entregado a través de Internet, proporcionando una disponibilidad "en cualquier sitio y a cualquier hora".

Por desgracia, durante los primeros años la calidad del contenido que se podía distribuir a través de los navegadores Web era muy pobre. Además, resultaba complejo *gestionar* el estado del aprendizaje de cada alumno: qué conceptos había comprendido ya, qué le faltaba por aprender, etcétera. Eso hizo que la industria de los contenidos educativos tardara en subirse al tren de las tecnologías Web, y, en los escasos puntos donde lo hizo de manera prematura, fue a través de soluciones basadas en ampliaciones (*plug-ins*) dependientes de cada fabricante.

Por otro lado, aunque fuera de manera manual, la Web permitió crear material directamente utilizando herramientas de diseño Web, no pensadas para generar contenido educativo. Con paciencia y un poco de cuidado, profesores y particulares podían crear sitios web con información sobre los más variados temas, e incluso donde se permitía una cierta interacción del usuario (como pequeños tests) mediante Java Script.

Todo esto ocasionó la aparición de gran cantidad de contenido educativo que no podía ser compartido. Por ejemplo, aunque alguien hubiera creado material para un curso sobre programación en C++, éste no podía utilizarse en un curso sobre programación de videojuegos por el uso de formatos diferentes. Es decir, la idea original del hipertexto para enlazar documentos relacionados, *no* podía realizarse entre el material educativo.

Ante este desorden, no es de extrañar que surgieran diferentes iniciativas de estandarización, como el LTSC (*Learning Technology Standard Group*) y el proyecto IMS (ambos de IEEE).

En 1.997, el Departamento de Defensa de Estados Unidos decide poner cartas en el asunto para buscar una solución a un método de enseñanza al que llevaban mucho tiempo mirando con buenos ojos. Con la intención de impulsar la adopción de tecnologías de enseñanza distribuida sofisticadas,

[24] `http://www.macromedia.com/software/authorware/`
[25] `http://www.toolbook.com/`
[26] `http://www.malted.com/`

funda la iniciativa ADL (*Advanced Distributed Learning*). Su primera misión fue intentar agrupar los diferentes estándares existentes. Se encontró con una amalgama terminológica muy confusa y con plataformas software (*middleware*) para la distribución de material educativo diversas e incompatibles. Esto significó que durante muchos meses los resultados fueran infructuosos, hasta que, en 1.999, publican el primer borrador de lo que denominaron SCORM (*Sharable Content Object Reference Model*). A pesar de resultar una propuesta aún muy inmadura, sirvió como catalizador y empujó a otros fabricantes y partes interesadas a aunar esfuerzos. En revisiones posteriores importaron ideas de otros grupos como AICC (*Aviation Industry CBT Committee*) e IMS (*Instructional Management System*) y generaron alrededor de SCORM una implementación de referencia y reuniones (*workshops*) donde discutir las ideas entre los diferentes implicados.

Hoy en día SCORM (Advanced Distributed Learning (ADL), 2004) se ha convertido en el estándar más importante para la generación, compartición y publicación de contenido educativo a través de la Web. Aunque tiene muchas partes, SCORM es un grupo de servicios para lanzar *contenido de aprendizaje* en un navegador, seguir la pista al rendimiento de cada alumno, intuir la mejor secuencia de presentación de contenidos, y controlar el grado de conocimientos de un alumno a lo largo de todo el proceso de aprendizaje. Todo este grupo de servicios es proporcionado por un LMS (*Learning Management System*, Sistema Gestor de Aprendizaje). El LMS proporciona el *sustrato* sobre el que se asienta el material del curso. Éste normalmente consiste en un grupo de páginas enlazadas, y SCORM entra en juego al facilitar la toma de decisiones acerca de cuando saltar de un sitio a otro. Normalmente los LMS son *aplicaciones Web* que se ejecutan en un servidor y que proporcionan el soñado acceso "en cualquier sitio a cualquier hora" al contenido de aprendizaje, pero al mismo tiempo haciendo relativamente sencilla la creación de dicho material por parte de profesores y educadores. En este contexto, lo que tradicionalmente se denominaban *marcos* (o frames) como unidades de contenido indivisible se llaman ahora *objetos de aprendizaje*, que suelen definirse como cualquier elemento que forme parte de la experiencia de aprendizaje y que pueda ser lanzado en un navegador. Naturalmente, a nivel de reutilización es preferible que sean pequeños y especializados.

Por otro lado, la llegada de Internet, aparte de la distribución de contenido, permitió el uso de nuevas herramientas para el aprendizaje colaborativo. El uso del correo electrónico, de *foros* (la evolución de las *news* y de Usenet) o de *wikis* facilita la colaboración entre gente situada a gran distancia. Hoy en día, los LMS permiten, aparte de la visualización de material educativo, hacer uso de muchas de estas herramientas, y todo administrado a través del mismo interfaz Web. Quizá los LMS comerciales más conocidos sean WebCT, Blackboard e IBM-Lotus Learning Management System. En 2.005 el primero fue comprado por el segundo, y aunque anunciaron su intención de mante-

ner los dos sistemas, también pronosticaron que a medio plazo crearían una nueva línea de producto fusionando en una única plataforma las mejores características de ambos. WebCT ha sido utilizado, entre otras muchas, en la Universidad Complutense de Madrid, en la Universidad de Barcelona, en la de Cádiz y en la UNED.

En el mundo del software libre (o código abierto) el número de alternativas es muy amplio. Entre ellas destacan Moodle, Claroline o Ilias.

El seguimiento de los estándares varía en cada una de ellas. Además, la facilidad de uso, las herramientas adicionales (foros, glosarios, etcétera), las posibilidades de ampliación o la forma de gestionar a los alumnos (importándolos de bases de datos externas, por ejemplo) hace que la elección entre ellas sea bastante compleja. No obstante, el uso de este tipo de sistemas se está imponiendo en la mayor parte de centros de enseñanza, especialmente en las Universidades. Los organismos oficiales comienzan a impulsar su uso por ser un método barato de proporcionar formación (aunque sea *no* reglada) a los ciudadanos. Por ejemplo, la Consejería de Educación de la Comunidad de Madrid ofrece un aula virtual con varios cursos que se distribuyen sobre la plataforma Moodle. Curiosamente, uno de ellos se enfoca precisamente al propio Moodle, para aquellos profesores que dispongan de él en sus centros educativos.

La educación Web es especialmente adecuada para enseñanza remota, de ahí que universidades que se basan en la educación a distancia como la UNED no hayan dudado en utilizarla. El uso como *apoyo* a la enseñanza reglada es también muy positivo. Dado que normalmente estos sistemas son utilizados por alumnos adolescentes y adultos, la crítica que se les hacía relativa a que sólo se centraban en la *dimensión intelectual* deja de tener sentido.

Una cosa que suele considerarse interesante en estos sistemas es la posibilidad de realizar exámenes. Como ya se ha ido esbozando, a lo largo del material del curso se realizan tests para comprobar que el alumno va comprendiendo cada concepto antes de pasar al siguiente. Pero *fuera de ese material*, muchos LMS permiten a los profesores crear baterías de preguntas para utilizarlas en exámenes que se realizan completamente de manera electrónica con el ordenador. Su validez es discutible si se realizan a distancia por la facilidad que los estudiantes tienen para engañar. No obstante, en clases presenciales resultan un método muy cómodo para hacer pruebas a los alumnos. Además, la propia plataforma las corrige y muestra diferentes estadísticas sobre los resultados. El profesor puede añadir comentarios pedagógicos a las respuestas *incorrectas* para proporcionar información instantánea a los alumnos sobre sus errores, algo que se considera muy positivo durante el aprendizaje.

Las diferentes plataformas suelen permitir crear las preguntas de manera *independiente* a los exámenes. Las preguntas pueden organizarse en categorías para luego crear exámenes de manera más sencilla. A modo de ejemplo,

Moodle soporta el siguiente tipo de preguntas:

- Verdadero o falso: una pregunta donde la respuesta es o bien verdadero o bien falso.
- Opción múltiple: una pregunta con opcionalmente una imagen, y varias respuestas, de las que una (o varias) podrían ser válidas.
- Respuesta corta: una pregunta y opcionalmente una imagen, para la que el alumno escribe en un cuadro de texto la respuesta. Ésta debería ser corta, ya que para corregirla se compara exactamente igual (opcionalmente sin tener en cuenta las mayúsculas) con una batería de posibles soluciones.
- Numérica: el resultado es un número. El alumno también ve un campo de texto que rellena, pero como es un número, para corregirlo es posible indicar un margen de error, de modo que si el profesor pone como respuesta correcta 30, con un margen de error de 5, si el alumno pone 27 se le considerará acertado. Se puede especificar una unidad para el resultado (por ejemplo metros), e incluso múltiplos de esa unidad (kilómetros, etcétera, junto con su multiplicador), y el sistema convierte las unidades adecuadamente para comprobar si la respuesta es correcta.
- Emparejamiento: tienes varias preguntas y varias respuestas, y el alumno debe emparejarlas.
- Preguntas incrustadas (*cloze*): es una pregunta compuesta de varias. Consiste en un texto en el que se pueden integrar listas desplegables con varias alternativas para que el alumno las elija, o con cuadros de texto para rellenar.
- Calculadas: son preguntas numéricas, pero en las que el profesor pone "huecos" que el sistema rellena automáticamente con valores, y calcula la solución a partir de una fórmula también proporcionada por el profesor. Por ejemplo, la pregunta podría ser "¿Cuanto es a + b?", y la respuesta a + b. El sistema elige al vuelo valores para ambas variables, determina el resultado y corrige la respuesta (numérica). Los posibles valores que se asocian a cada variable de la respuesta también pueden especificarse.

Moodle mantiene de manera separada las preguntas y los exámenes. Es posible disponer de una base de preguntas a partir de las que montar los exámenes concretos en cuestión de minutos. Cuando se está creando un examen, es posible utilizar dos tipos adicionales de "preguntas":

- Aleatoria: el sistema elige una pregunta aleatoria de una categoría elegida por el profesor. De esa forma, se pueden hacer muchas preguntas,

y poner un examen con 10 preguntas aleatorias. Cada alumno realizará un examen diferente.

- Emparejamiento de respuesta corta aleatoria: la idea es especificar una categoría, y el propio sistema crea la pregunta. Para eso, coge varias preguntas del tipo "respuesta corta", junto con su respuesta, y las junta en una pregunta del tipo "emparejamiento", de modo que el alumno debe responder correctamente a varias preguntas pero viendo las respuestas de todas ellas. Al ser aleatoria, si el examen se repite, esta pregunta será diferente.

Por desgracia, la especificación de las preguntas y los exámenes es una característica que *queda fuera* del propio contenido, que es lo que SCORM se encarga de estandarizar. El resultado es que cada LMS proporciona unos tipos de preguntas diferentes, y resulta complejo importar baterías de preguntas de un sistema a otro.

2.7.4. Desventajas de los sistemas basados en marcos

Este tipo de sistemas ha conseguido una gran difusión, especialmente con la aparición de la Web y la llegada de los estándares de creación de material. Quizá las causas sean la "facilidad" para crear ese material, lo que abarata los costes, y la interacción sencilla que no requiere demasiado tiempo de aprendizaje por parte de los usuarios.

Por desgracia, sufren algunos inconvenientes. Éstos pueden clasificarse en dos grandes áreas. La primera de ellas se refiere a, digamos, *ingeniería*. Estos sistemas obligan a los profesores a proporcionar completamente el texto a presentar, todas las preguntas y sus respuestas asociadas, y a mantener un control estricto del flujo del curso, permitiendo (en el mejor de los casos), especificar diferentes ramas dependiendo de las respuestas (prefijadas) de los alumnos. Tal y como dice Rickel (1989), los ordenadores son utilizados aquí como pasadores de página electrónicos.

Con los exámenes ocurre algo parecido. Aunque algunas herramientas permiten una cierta autonomía en la generación aleatoria de preguntas, en la práctica el profesor tiene que preocuparse de plantear un gran número para que el alumno se enfrente a ejercicios nuevos en cada iteración. También se han realizado algunas mejoras como el llamado "CAI generativo" (*generative CAI*) que utiliza plantillas de texto y gramáticas de generación de problemas para mejorar la creación automática de problemas, pero su uso no es siempre generalizable.

Esto nos acerca a la segunda gran pega: la pedagogía. Aunque consigamos mejorar la generación automática de ejercicios, éstos serán siempre *aleatorios*, es decir, no se *adaptan* a los conocimientos y carencias de cada alumno. Feifer (1992) dice que, al fin y al cabo, en los sistemas de enseñanza basados en

marcos el ordenador recibe un *guión* (*script*) para dirigir la interacción con el estudiante. Independientemente de que se permitan guiones ramificados, éstos serán siempre una combinación *anticipada y prevista por el profesor* de las respuestas del alumno. Por tanto, la interacción sólo será apropiada si las respuestas encajan con este guión. Si un estudiante desea dar una respuesta, hacer una pregunta, o profundizar en material no previsto por el creador del curso, se verá atado a un plan que no quiere seguir.

La sencillez inherente de la interacción también limita el tipo de material que puede enseñarse. Aunque las alternativas de las preguntas soportadas por Moodle puedan parecer amplias, es fácil caer en la ambición y preguntarse si podríamos utilizar este tipo de herramientas para, por ejemplo, *enseñar* a un niño a sumar (y dar *explicaciones* contextualizadas a sus errores en el uso del acarreo por ejemplo), a un técnico a arreglar calderas, o a un militar a pilotar submarinos.

El resto del capítulo enumera otros tipos de sistemas educativos que intentan mejorar en estos aspectos, normalmente a costa de incrementar la complejidad del software desarrollado.

2.8. Simuladores

El mayor punto débil de los sistemas basados en marcos de la sección anterior se encuentra en sus propias raíces: el conductismo. Como se ha dicho, esto ocasiona interacciones relativamente sencillas, donde las posibilidades pedagógicas quedan sesgadas en muchos dominios.

Por ejemplo, no cabe duda de que una parte importante de muchos procesos de aprendizaje es el *entrenamiento*. Aunque una cierta base teórica siempre es importante, el periodo de aprendizaje no se puede considerar completamente terminado si los alumnos no pasan por una fase en la que se pongan en practica las habilidades que deben aprender.

Estas sesiones de entrenamiento pueden realizarse con profesores que asisten a los estudiantes y solucionan los problemas que aparezcan y para los que el alumno aún no está preparado. Este modo de enseñanza, mediante un *mentor*, ha sido utilizada durante siglos.

El avance de las tecnologías ha originado la aparición de tareas más especializadas que requieren entrenamientos más específicos, y con más riesgos. Claros ejemplos de esto son los entrenamientos de pilotos noveles en avionetas, de conductores de trenes, de capitanes de submarinos, y un largo etcétera. En todos estos casos, la práctica mediante un *mentor* sigue siendo posible, pero tiene ciertos problemas. Durante el entrenamiento, se ponen a disposición del alumno sistemas generalmente muy caros, que corren el riesgo de resultar gravemente dañados al ser manejados por manos inexpertas. Aún más grave es el riesgo de causar daños humanos por culpa de accidentes que el tutor no sea capaz de evitar (por ejemplo, causados por aspirantes

Figura 2.3: Simulador de vuelo artesanal de principios del siglo XX

a controladores aéreos monitorizando el tráfico de un gran aeropuerto). En estos casos, el entrenamiento en entornos reales resulta por lo tanto muy caro y arriesgado.

Dado que el primer dominio en el que surgió este problema fue en la aviación, no es de extrañar que fuera en ese ámbito en el que surgieron los primeros *simuladores* donde el alumno puede entrenar y practicar sus habilidades de pilotaje sin ningún riesgo. Hoy, en un mundo digital, tenemos una idea sesgada en este aspecto. En realidad los primeros "simuladores" surgieron muy a principios del siglo XX, antes de la aparición de los ordenadores. Los sistemas fabricados eran, por lo tanto, completamente mecánicos, dedicados a realizar una *imitación empírica* de los efectos del vuelo. De hecho, en los primeros años esa imitación era completamente artesanal, recreada manualmente por los instructores (figura 2.3).

Page (2000) cuenta que, como suele ocurrir, las necesidades militares impulsaron las principales mejoras en aquella época. La Primera Guerra Mundial exigió un número relativamente alto de pilotos, que necesitaban instrucción, por lo que hubo que automatizar el funcionamiento, para que la imitación no fuera recreada manualmente por instructores sino gracias a sistemas *mecánicos* automáticos. Por su parte, la Segunda Guerra Mundial hizo un uso aún más importante de la aviación, lo que creó nuevos retos. Afortunadamente en aquel momento la Informática comenzaba tímidamente a aparecer, por lo que se integraron de manera esporádica en los simuladores mecánicos anteriores los por aquel entonces conocidos como *analizadores diferenciales* para *simular* por primera vez la respuesta aerodinámica de los aviones.

No obstante, esa simulación seguía siendo poco precisa por la escasez

de datos. Además, la generalización de los vuelos comerciales empujó a los simuladores a usos intensivos (llegando incluso a ser utilizados los siete días de la semana) lo que ponía a prueba sus componentes mecánicos.

No fue hasta la década de 1.950 cuando los diferentes fabricantes de aviones comenzaron a tomarse en serio la recogida de datos respecto al comportamiento físico de sus diseños. Junto con la aparición de la informática digital, esto desembocó en la proliferación de una nueva generación de simuladores que abandonaban por fin los sistemas analógicos. En aquel momento, los ordenadores disponibles no tenían la potencia suficiente como para soportar las necesidades de cálculo que un simulador físico de vuelo requería, por lo que a menudo se creaban computadores pensados específicamente para ello.

A nivel del desarrollo de software, inicialmente se realizaba, como en otros ámbitos de la Informática, de manera artesanal. Nance (1983) cuenta que en 1.960 se identificaron por primera vez las funciones que aparecen de forma reiterada dentro del desarrollo de modelos, creando el bautizado como *General Simulation Program* (GSP) que convenció al mundo de la importancia del software (y no sólo del hardware y los modelos físicos) dentro del ámbito de los simuladores. También sirvió como pistoletazo de salida a un sin fin de metodologías e intentos de estandarización para su desarrollo. Quizá el legado más importante (o al menos más ampliamente difundido) de toda aquella investigación es la *orientación a objetos*, como resultado del lenguaje SIMULA (Nygaard y Dahl, 1978; Holmevik, 1994). La primera versión del lenguaje, conocida como SIMULA I y desarrollada entre 1.964 y 1.965, estaba enfocada al desarrollo de simuladores de eventos discretos. El nombre, de hecho, se creó como acrónimo de ***Simu****lation* ***La****nguage*. La segunda (y más extendida) versión data de 1.967[27] convirtiéndose en un lenguaje orientado a objetos general que soportaba de manera nativa listas enlazadas y simulación orientada a procesos discretos. Curiosamente, en este punto cambiaron el significado del nombre a ***Sim****ple* ***U****niversal* ***la****nguage*.

Por otro lado, la *visualización* avanzó de manera paralela. Inicialmente se pretendía que el piloto experimentara de una manera somera las reacciones motrices del avión. Para conseguir que los aprendices tuvieran una realimentación *visual* de algo parecido a volar tuvo que pasar más tiempo. Según Vince (1995), durante mucho tiempo se utilizaron maquetas de aeropuertos que eran recorridas por una pequeña cámara en tiempo real imitando el movimiento del avión que se estaba simulando en función de las órdenes del estudiante.

La calidad visual de estos sistemas no era demasiado realista; además al ser dependiente de una *maqueta física* resultaba poco viable (o muy caro)

[27] De ahí que a veces se la conozca como SIMULA 67, aunque hoy en día debido al abandono de SIMULA I, ha dejado de ser necesario distinguir entre ambas y se habla sencillamente de SIMULA.

Figura 2.4: Simulador de un Boeing 747 de la NASA

disponer de diferentes entornos (aeropuertos) donde entrenar. Para solventar esos problemas, algunos simuladores se basaban en *vídeos*, que eran reproducidos a diferente velocidad en función de la velocidad de aproximación seleccionada por el piloto en el simulador. Este tipo de sistemas eran mucho más pobres, aunque tuvieron cierta repercusión en algunos dominios, como en la conducción de trenes donde el número de "ejes de libertad" del conductor es mucho más reducido.

Desde la llegada de los gráficos generados por ordenador en tiempo real el panorama ha cambiado completamente. Las maquetas y vídeos han sido sustituidos por representaciones virtuales de los aeropuertos, que son utilizadas por sistemas informáticos para generar las imágenes en tiempo real (figura 2.4). El alumno sigue manejando el sistema a través de un cuadro de mandos semejante al de, por ejemplo, un avión real, pero las imágenes visualizadas son generadas completamente por ordenador. Estos sistemas heredan la ventaja de eliminar los riesgos de daños en material o en seres humanos, que ya tenían los simuladores anteriores. Además, continúa Vince, aporta nuevos beneficios:

- *Precisión*: los modelos virtuales utilizados pueden construirse a partir de entornos reales con una gran precisión. Por ejemplo los aeropuertos pueden ser añadidos a los simuladores a partir de los planos realizados por los arquitectos mediante herramientas CAD (*Computer Aided Design*, Diseño asistido por ordenador). Además, la actualización de los modelos debido a, por ejemplo, obras de mejora puede realizarse rápidamente sin demasiados problemas. Por último para un más efec-

tivo entrenamiento es deseable disponer de distintos entornos donde ensayar (por ejemplo diferentes aeropuertos, tramos de vía, etc). Los sistemas anteriores, como ya se ha mencionado, requerirían una maqueta por cada uno de esos entornos, lo que limita su disponibilidad. Al utilizar entornos virtuales, contar con distintos modelos digitales tridimensionales de esos entornos resulta mucho más viable.

- *Interacción*: la interacción entre el entorno y el sistema simulado puede mejorarse. Por ejemplo, las colisiones en los simuladores basados en maquetas resultaban un problema, pues era necesario detectarlas, pero hacerlo de forma física podía dañar la maqueta. Además el tamaño de las cámaras que recorrían dichas maquetas podían restringir sus movimientos. Al utilizar entornos sintéticos la detección de colisiones puede realizarse sin los problemas anteriores. Además el sistema es capaz de generar imágenes desde cualquier punto de vista, pues la "cámara" puede colocarse en cualquier posición.

- *Entornos mejorados*: los sistemas de simulación de este tipo no están limitados a la representación de entornos estáticos por donde se desplaza el vehículo, tal y como ocurría en el caso de las maquetas. Ahora es posible añadir nuevos elementos dinámicos que se desplazan libremente por el entorno, como pueden ser otros aviones, trenes, peatones, coches, etc.

- *Niveles de luz y condiciones atmosféricas*: al generar las imágenes de forma sintética es posible modificar ciertos parámetros del dibujado (*renderizado*) para adaptar la simulación. De ese modo es posible simular diferentes horas del día o estaciones del año, e incluso distintas condiciones atmosféricas (sol, lluvia, nieve, niebla...) de modo que el estudiante puede practicar en todas ellas.

- *Condiciones extremas*: quizá más interesante aún que la anterior es que los sistemas informáticos permiten el entrenamiento de situaciones extremas. Estas situaciones son difíciles de simular en otro tipo de sistemas, y resulta inviable (debido a su alto riesgo) practicarlas en el mundo real en los entrenamientos con mentor. Sin embargo el personal especializado debería estar preparado para hacer frente a este tipo de situaciones, como por ejemplo aterrizajes de emergencia en lugares poco aptos (autopistas, prados, montaña, pistas heladas...) ante condiciones meteorológicas adversas (nieve, niebla...) y con deficiencias en los vehículos (fallos en motores, reventones en el tren de aterrizaje...).

  A pesar de tanto "dramatismo" esta ventaja es aplicable en situaciones más cotidianas. En simuladores de conducción para autoescuelas, es posible conseguir que el alumno practique situaciones poco comunes, como cambios en las carreteras, en las señales, modelos de vehículos

> nuevos, accidentes, etc. Incluso con este tipo de sistemas es posible analizar la repercusión de ciertos cambios en el entorno antes de realizarlos, o llevar a cabo estudios de todo tipo, como si los carteles de propaganda que solía haber en las carreteras realmente suponían una distracción a los conductores (Leitão et al., 1999).

En resumen, las ventajas principales de estos sistemas son la eliminación de riesgos, con la consiguiente bajada de costes, y el permitir el entrenamiento bajo una gran cantidad de diferentes condiciones.

Para poder realizar estos simuladores, es necesario *disponer de un modelo* de aquello que se simula para poder *imitarlo con precisión*. De ahí la importancia de la toma de datos objetiva que en el mundo de la aeronáutica se hiciera patente a mediados del siglo pasado. Naturalmente, la adquisición de esos datos no significa de manera directa la posibilidad de simularlos. En los primeros años de la informática digital, la creación de buenos simuladores era un gran reto, lo que, como ya se ha mencionado, generó un área muy activa de investigación.

La dificultad inherente tiene, sin embargo, una ventaja que hace que el esfuerzo merezca la pena: el concepto de *simular* gracias a la existencia de un *modelo* es increíblemente poderoso. En el caso del dominio de la aviónica los simuladores surgieron como una *alternativa* al entrenamiento, pero en otros ámbitos fueron usados como *la única posibilidad*. Es reconocido universalmente que habría resultado imposible poner un hombre en la Luna si no hubiera sido gracias a los simuladores, que eran la única forma de entrenamiento para el viaje espacial. También resulta legendario el uso de la simulación para ayudar a resolver los problemas de la misión del Apolo 13.

Esto demuestra que la disponibilidad de modelos permite usos de los simuladores menos obvios inicialmente. En el caso de la carrera espacial, los simuladores se utilizaron para practicar cosas *que no estaban al alcance* en la realidad.

Las posibilidades son inmensas. Una vez que se dispone de un modelo, resulta relativamente sencillo *experimentar con el tiempo y las escalas*. Es posible acelerar el tiempo para, por ejemplo, realizar un simulador macroeconómico donde las semanas pasan a la velocidad de segundos, o ralentizarlo para mostrar el movimiento de datos producido en el interior de un procesador. Jugando con las escalas espaciales, podemos enseñar el movimiento planetario (obviamente, acelerando también el tiempo), o mostrar la interacción entre diferentes moléculas de agua (Trindade et al., 1999). En este sentido, el uso de los simuladores puede superar el mero interés educativo para pasar a un plano de investigación y predicción. Por ejemplo, asumiendo conocido el modelo, es posible simular el *big bang*, el cambio climático, o la evolución del agujero de ozono.

Volviendo a los usos educativos, la posibilidad de acelerar el tiempo per-

mite, por otro lado, hacer explícitas las repercusiones globales de las decisiones. Esto facilita que los estudiantes pasen de una visión analítica del área que están estudiando (en un plano teórico) a asimilar de manera intuitiva los resultados de sus acciones (desde un punto de vista práctico).

Por último, gracias a los simuladores y a la existencia de modelos podemos recrear cosas que no están a nuestro alcance (como el simulador de vuelo espacial en las fases iniciales del proyecto Apolo), o incluso *que no existen* o desaparecieron tiempo atrás. Así podemos recrear una sociedad antigua para mostrársela a los alumnos o simular un mundo sin rozamiento para que aprendan el principio de la conservación de la energía (Dede et al., 1996).

En resumen, tal y como comenta Jane Boston en una entrevista para Aldrich (2005), los simuladores pueden ser utilizados de manera acertada de cuatro maneras:

- Para que los alumnos desarrollen una comprensión profunda de las *grandes ideas y conceptos* (aquellas para las que sólo la experiencia puede conseguir un entendimiento profundo). A modo de ejemplo, comenta que no es lo mismo aprender de memoria que un ecosistema es frágil, a formar parte de uno en el que tus decisiones afectan a los demás seres vivos.

- En el ejemplo anterior, es necesario acelerar la velocidad a la que transcurre el tiempo para poder mostrar los resultados de nuestras acciones durante una sesión. En ese sentido, los simuladores son ideales porque permiten jugar con el tiempo y las escalas a voluntad.

- También son adecuados para aprender y practicar a tomar decisiones antes de enfrentar al estudiante a situaciones reales críticas donde se pone en juego material de alto coste o incluso vidas humanas.

- Por último, son ideales para poder situarnos en lugares o momentos de la historia en los que es imposible o improbable que podamos estar físicamente.

También Aldrich nos cuenta (página 78) que la fidelidad de una simulación depende de la precisión y exactitud con la que imite al original que se recrea. Además, las buenas simulaciones son útiles porque en general la práctica hace mejor a la gente en aquello a lo que se dedican[28].

En realidad aquí surge en cierto modo una incoherencia: si lo importante es practicar, ¿resulta tan necesaria la exactitud? Naturalmente, cierta

[28]Como ejemplo, nos cuenta que las estadísticas demostraron que en la Segunda Guerra Mundial, un piloto tenía muchísimas más probabilidades de sobrevivir a su siguiente combate aéreo cuanto mayor fuera el número de batallas en las que había participado anteriormente (página 82).

fiabilidad es necesaria para evitar aprender cosas que luego no ocurren en la vida real, pero, ¿tan crítico es el propio *modelo*? Como suele ocurrir, la respuesta dependerá del dominio. Crear un modelo razonablemente preciso del funcionamiento de un avión seguramente sea necesario para un correcto entrenamiento. Pero crear un simulador preciso del funcionamiento de la economía a nivel mundial resulta impráctico: en realidad *no* conocemos sus reglas exactas, por lo que resulta imposible realizarlo. En esos casos, podemos querer seguir haciendo simuladores donde el "modelo" se implementa de manera mucho más artesanal, como por ejemplo utilizando una serie de reglas básicas en lugar de utilizando un conocimiento exacto (que no tenemos) sobre el dominio concreto. En este sentido, Aldrich también revisa de manera somera diferentes métodos de implementaciones *ad hoc*, como por ejemplo las que conoce como *branching stories* (sospechosamente parecidas a los sistemas basados en marcos) y los sistemas "basados en hojas de cálculo".

2.9. Tutores inteligentes

2.9.1. Introducción

Los sistemas de enseñanza asistida basados en marcos que se describieron en la sección 2.7 estaban fuertemente arraigados en las teorías conductistas del aprendizaje. Disponer de dichas teorías como inspiración para su construcción proporciona una indudable ventaja práctica a la hora de diseñar el material. Además, según McKenna y Laycock (2004), las aplicaciones resultantes disponen de interfaces claros que proponen interacciones a los que la mayor parte de los usuarios están acostumbrados.

Sin embargo, existen dos puntos de mejora que, *en principio*, resultan independientes:

- *Acercamiento al cognitivismo*: en muchos dominios el conductismo ha demostrado ser insuficiente, por lo que una evolución natural, aun a costa de añadir complejidad en la interacción entre el usuario y el sistema, es tender hacia el cognitivismo. Para eso, hay que abandonar la idea de alternar explicaciones teóricas y ejercicios que potencien el recuerdo y las respuestas condicionadas para comenzar a dar una mayor importancia al propio alumno y a sus actividades. Este es el camino seguido por los simuladores vistos en la sección 2.8.

- *Mejora en la implementación*: el modo en el que se desarrollan los sistemas basados en marcos resulta increíblemente artesanal. A pesar del soporte proporcionado por las cada vez más sofisticadas herramientas de autoría, no deja de ser cierto que el *creador del contenido* (la parte vital de todo sistema de enseñanza) se enfrenta a una ardua labor de análisis de los diferentes errores habituales y sus posibles causas.

Aunque esta separación parezca mostrar que la aparición de los simuladores no es más que una evolución natural a partir del análisis de los problemas de los sistemas basados en marcos, en la práctica esto *no* ocurrió así. Las fechas proporcionadas en las dos secciones anteriores demuestran que, de hecho, ambos mundos evolucionaron de manera paralela.

Sin embargo, sí fue un análisis de este tipo el que pudo ocasionar en 1.970 la aparición de una nueva rama en los sistemas de enseñanza. Esta separación en el modo de implementar los sistemas buscaba ahorrar trabajo al tutor a costa de unos mayores requisitos en el *hardware*. Increíblemente, como veremos enseguida, terminó también alejándose del conductismo y acercándose de manera creciente hacia el cognitivismo.

Para explicarlo, me inspiraré en un ejemplo mostrado por Ong y Ramachandran (2000). Asumamos que queremos realizar un sistema de enseñanza basada en marcos para que un grupo de alumnos de primaria aprendan y practiquen la suma. En principio, por medio de texto, dibujos e incluso animaciones coloristas adaptadas a su corta edad se les explicaría cual es el proceso de realizar sumas sencillas. Además, los alumnos tendrán que realizar multitud de ejercicios, dado que a sumar se aprende sumando.

El problema en la parte de preguntas al que nos enfrentaríamos en el caso de utilizar un sistema basado en marcos es la gran cantidad de posibilidades de *equivocarse*. Debido al modo de funcionar del modelo subyacente de este tipo de sistemas, estaremos obligados a poner una gran cantidad de opciones posibles para que los niños elijan la que ellos crean correcta. Desde el punto de vista del creador de contenido (ejercicios), esto supone que deberá anticiparse a todos los posibles errores típicos que los niños pueden cometer. Y aun así, seguramente se dará el caso de que algún niño *no* encontrará la opción que él había considerado como válida, replanteándose automáticamente su respuesta y volviendo a intentar la suma (o, peor aún, eligiendo una aleatoriamente). O podría darse el caso de que los niños ni siquiera intentaran *realizar* la suma, sino sencillamente deducir el resultado a partir de las respuestas propuestas. En realidad este último problema ya lo reconocía Skinner (1958), al comentar que un test potencia *el reconocimiento* en lugar *del recuerdo*; sin embargo también argumentaba que en aquel momento los tests eran mucho más fáciles de implementar (o, de hecho, la única opción posible).

Naturalmente, enseguida se dispara la alternativa de que, en lugar de ser el tutor quien coloque las respuestas, sea el propio sistema. Después de todo, seguramente lo que mejor sepa hacer un ordenador será sumar, por lo que no le resulta nada difícil saber si el alumno ha acertado o no. El problema es que si realmente queremos *enseñar*, el sistema debería proporcionar explicaciones que intenten aclarar a los niños las causas de los errores para que aprendan de sus propias equivocaciones. En el modelo basado en marcos esto impone aún más trabajo para el creador de ejercicios, porque no sólo tendrá que

	22	46
Estudiante A:	+39	+37
	51	73
	22	46
Estudiante B:	+39	+37
	161	183
	22	46
Estudiante C:	+39	+37
	62	85

Tabla 2.1: Respuestas de tres alumnos a dos ejercicios de suma

pensar posibles formas de equivocarse para cada pregunta, sino que además tendrá que añadir frases explicativas a cada una de ellas para que el sistema las proporcione en el caso de que se seleccionen.

Por tanto, el principal problema estriba en que los guiones (o *scripts*) en los que están basados en última instancia los sistemas basados en marcos (Feifer, 1992) *no escalan bien*. En realidad, continúa Feifer, los profesores humanos *no* responden a las preguntas o dan ayuda a los alumnos en función de un guión. En lugar de eso, disponen de conocimiento sobre el área que enseñan, tienen una intuición sobre lo que sabe (o debería saber) el alumno, y tienen conocimientos sobre estrategias para enseñar. Esas tres fuentes de conocimiento se utilizan de manera conjunta para determinar en cada momento cómo interactuar con el alumno para conseguir que aprenda. Dicho de otra forma, los sistemas basados en marcos limitan el potencial de los ordenadores, dado que suelen generar comportamientos estructurados y dirigidos de antemano (Inoue, 2000).

A modo de ejemplo, asumamos el escenario mostrado en la tabla 2.1, donde se enfrenta a tres alumnos diferentes a las mismas dos sumas, y los tres se equivocan en ambas. Lo significativo es que aplicando el sentido común sobre el *algoritmo de suma* (con lápiz y papel) no resulta difícil darse cuenta de la causa *profunda* de los errores:

- *Estudiante A*: nunca "acarrea", es decir tiene problemas con lo que en primaria nos enseñaron que se llamaba *sumar llevando*.
- *Estudiante B*: tiene el problema opuesto, dado que *siempre* acarrea.
- *Estudiante C*: aparentemente comprende los pasos de "llevarse uno", pero tiene problemas con la suma de dígitos individuales.

Naturalmente, para implementar un sistema educativo basado en marcos

que fuera capaz de dar esas explicaciones, el creador de los ejercicios debería, como ya se ha dicho, haber previsto este tipo de errores y haber incluido las tres respuestas erróneas para cada una de las sumas. La pregunta que asalta es si es posible añadir al sistema la capacidad de "saber" sumar de la misma forma que nosotros para que "razone" tal y como lo acabamos de hacer y obtenga *de manera automática* esas tres mismas conclusiones de las causas de los errores. Obviamente, hacer un sistema con esta capacidad es mucho más complicado que implementar un programa que se limita a interpretar un guión[29]. El peso del desarrollo se traslada desde la mera autoría de enmarañados y repetitivos ejercicios hacia la *programación* del sistema. Pero, de conseguirlo, habremos logrado que las uniones entre las respuestas y sus resultados (con posibles explicaciones) se puedan hacer ahora "al vuelo", en lugar de estar cableadas desde el principio.

Al añadir información sobre *cómo sumamos* se dice que hemos enriquecido a nuestro sistema con *conocimiento del dominio*. En principio, este modo de hacer evolucionar *la implementación* de los sistemas basados en marcos *no* añade nada nuevo "al modo de educar"; podríamos seguir usando las ideas del conductismo, y haciendo que el sistema genere varias respuestas incorrectas a las preguntas. En ese caso, la única ventaja está en el ahorro en tiempo en la generación de ejercicios.

Sin embargo, una vez que tenemos el conocimiento del dominio, resulta bastante fácil dejar volar la imaginación y utilizarlo de maneras más originales, al igual que ocurría cuando se dispuso de modelos en el mundo de los simuladores. Por otro lado, las primeras apariciones de esta nueva idea de implementación surgieron para dominios en los que, efectivamente, los guiones de los sistemas basados en marcos no escalaban lo suficientemente bien y se necesitaba otra alternativa diferente que evitara tener que hacer las uniones de los marcos a mano.

Eso ocurrió en dominios en los que la resolución de los ejercicios *no era tan trivial* como dar una respuesta, sino que los alumnos tenían que pensar más, dar varios pasos en los que podían equivocarse (ocasionando así una explosión combinatoria en el número de posibles respuestas incorrectas y sus causas) y haciendo las preguntas tipo test poco efectivas.

Lo verdaderamente gracioso de esto es que en realidad en esos dominios el principal problema estaba en las propias *teorías conductistas*. Por ejemplo, resulta imposible enseñar a encontrar errores en un circuito electrónico mediante *drill-and-practise*; necesitamos añadir a los sistemas informáticos los razonamientos que hace un experto cuando intenta dar una explicación racional a los errores de su pupilo.

Aquí se ha descrito la llegada a los sistemas con conocimiento del dominio como una evolución natural debido a la, en algunos casos, costosa creación

[29] La historia de la Inteligencia Artificial tristemente nos recuerda que los ordenadores no muestran de manera natural razonamiento de sentido común.

de material en los sistemas basados en marcos. Pero, curiosamente, añadir "razonamiento" a los sistemas significó un desplazamiento hacia el cognitivismo en la enseñanza asistida, no porque ese razonamiento no pudiera usarse (quizá) en sistemas conductistas, sino porque sólo merecía la pena el esfuerzo en los dominios donde la enseñanza debía ser cognitivista. De esa forma, tal y como nos dicen Frasson y Aimeur (1996) estos nuevos sistemas implican una participación por parte de los alumnos mucho más activa en su propio aprendizaje.

Este cambio en el paradigma de la enseñanza resulta, en realidad, bastante claro en el ejemplo de nuestro sistema para enseñar a sumar, si nos damos cuenta de lo que realmente está haciendo el programa. En el caso de un sistema conductista lo que queremos es probar que realmente hemos conseguido ocasionar en el estudiante un *cambio en la conducta* que hace que efectivamente proporcione las respuestas (condicionadas) a las preguntas que le hacemos, sin que necesariamente se entienda la razón[30]. Desde el punto de vista del cognitivismo, sin embargo, las preguntas y respuestas nos sirven para asegurarnos de que el conocimiento que el alumno ha creado a partir de la información suministrada es correcto, y no sufre *errores de reconstrucción*. Así, no buscamos comportamientos automáticos, sino razonados. En este sentido, las posibles causas de equivocación, incluso en un dominio tan simple como la suma, pueden ser muchas. De ahí que aspirar a tener todo cableado resulte totalmente impráctico y resulte más interesante intentar "meter un profesor en un ordenador", que *razone* sobre los posibles *errores de reconstrucción* del alumno.

Todo esto demuestra lo que se comentaba al principio de que la búsqueda de un modo mejor en el modo de implementar los sistemas de enseñanza basados en marcos terminaría desembocando en un abandono del conductismo. Esto los llevó, de hecho, al paradigma seguido por los simuladores, el cognitivismo. Y es que, en el fondo, al añadir conocimiento del dominio tenemos *un modelo* del área que enseñamos al igual que en los simuladores. Después de todo, los dos puntos de mejora con los que iniciábamos la sección no están tan alejados como pensábamos en un principio. La principal diferencia es que los simuladores *sólo proporcionan práctica*, pero no explicaciones respecto a lo que el alumno ha hecho bien o mal. En este sentido, los simuladores como tal (sin apoyo externo de tutores humanos que realicen un análisis del resultado) proporcionan *educación informal y dinámica*[31], mientras que este nuevo tipo de sistemas educativos sí dan ayuda y potencialmente una selección cuidada de los ejercicios por lo que saltan al campo de la educación lineal.

[30] Esto puede ser una visión un tanto extrema del conductismo, pero nos sirve como ejemplo.

[31] Estos términos se definieron en la página 19.

2.9.2. Definición de los Tutores Inteligentes

Como ya hemos dicho, al desplazar el peso del desarrollo desde la creación del contenido hacia la implementación, la programación resulta mucho más compleja, y la carga en el *hardware* también se hace más patente. Debido a esto, y tal y como nos dice Lowe (2001), las teorías conductistas influyeron en los primeros sistemas de enseñanza asistida, y sólo cuando la tecnología mejoró, el software comenzó a mirar con mejores ojos las teorías cognitivistas.

Pero en cualquier caso, ¿cuales son las diferencias principales entre las dos aproximaciones para la creación de sistemas de enseñanza? Ya vimos en la sección 2.7 que en realidad los sistemas basados en marcos equivalen a "libros" como contenedores del conocimiento del autor del contenido. Éste utiliza las herramientas básicas de un libro (como líneas, párrafos, capítulos, figuras, índices, índices por palabras...) para plasmar lo que sabe y la que considera mejor forma de transmitirlo. Naturalmente, estos sistemas también utilizan otras ideas más sofisticadas con las que Thorndike sólo podía soñar, como enlaces entre capítulos, partes que se pueden saltar y son pospuestas para más adelante, preguntas, etcétera. Pero en cualquier caso, *todo esto debe estar previsto de antemano.* El creador del contenido debe plasmar su conocimiento sobre el área que enseña, y también sus ideas pedagógicas sobre las mejores formas de exponerlo, como por ejemplo en el orden de la exposición. Por tanto, se dice que estos sistemas *codifican decisiones* sobre el conocimiento pedagógico y del dominio. Este modo de proceder se asienta en que los ordenadores son malos en la improvisación, pero superan a los humanos en precisión y completitud en la información de la que disponen o, dicho de otra manera, un ordenador (y un libro) tienen mucha más paciencia y memoria que una persona.

Naturalmente esto no significa, en absoluto, que crear el contenido para uno de estos sistemas sea una tarea trivial. Aparte de las teorías pedagógicas, hay mucha investigación en cómo reutilizar material, cómo hacer herramientas que faciliten su construcción, estándares, etcétera, tal y como quedó reflejado anteriormente.

Con la nueva aproximación, en lugar de *guardar las decisiones* pasamos a *codificar el conocimiento.* En realidad los sistemas basados en marcos también mantienen "conocimiento", pero está *cableado* en las decisiones, por lo que podemos verlo como *conocimiento procedimental*, que dice qué hacer en cada caso. La alternativa es incluir ese conocimiento (del dominio y pedagógico) de manera declarativa con el que *las aplicaciones puedan razonar* sobre qué paso es el más propicio en un determinado momento.

Esta idea fue propuesta por primera vez por Carbonell en 1970. Comparaba los sistemas de enseñanza basados en marcos (las "unidades de conocimiento" creadas por los diseñadores) con su nueva alternativa *orientada a la información estructurada.* Como se ha dicho, dio por primera vez una solu-

ción a la necesidad de anticiparse a todas las respuestas del usuario impuesta por los marcos que necesitan tener registrado por dónde continuar en cada caso. La solución, naturalmente, era el uso de técnicas de Inteligencia Artificial mediante una representación explícita y declarativa del conocimiento. Durante mucho tiempo a estos nuevos sistemas se les conoció con las siglas ICAI (*Intelligent Computer Assisted Instruction*). No obstante resultó obvio que las repercusiones de este nuevo modo de implementación eran demasiado grandes (tanto a nivel del desarrollo como del resultado final) como para que sólo se diferenciaran en una letra adicional del nombre. Quizá debido a ello, Sleeman y Brown acuñaron un nuevo acrónimo en su libro de 1982, pasando desde entonces a ser conocidos como *Intelligent Tutoring Systems* (Sistemas Inteligentes de Tutoría) o, simplemente, ITS. Hoy, este término se utiliza, por lo tanto, para referirse a cualquier programa utilizado para enseñar y que contiene "inteligencia", algo tan general que ha ayudado a hacer el área de investigación en ITS increíblemente grande y variada (Freedman et al., 2000).

En este sentido, Kopec y Thompson (1992) nos dicen (página 94) que el campo de los ITS ha realizado análisis detallados sobre los procesos de aprendizaje (correcto o erróneo) y enseñanza, ha implementado diferentes paradigmas basados en las teorías de la pedagogía, y ha empleado y desarrollado teorías y técnicas de Inteligencia Artificial con el objetivo de mejorar el funcionamiento general de los ITS.

Esto pone de relieve que el avance de los ITS ha ido de la mano del avance de la Inteligencia Artificial, campos que han bebido el uno del otro en una colaboración estrecha. Muestra de ello es que el sistema propuesto por Carbonell que inició el campo[32] se asentaba en las *redes semánticas*, que habían aparecido por primera vez sólo dos años antes. Carbonell pronosticó que con ellas se podrían crear sistemas que mantuvieran diálogos de iniciativa mixta (es decir, comenzados indistintamente por el tutor o por el alumno) al igual que ocurre en las clases (al menos individuales) reales.

Con el tiempo, las redes semánticas mostraron sus debilidades (al ser incapaces de mantener conocimiento procedimental y herencia de propiedades), y fueron abandonadas. Desde entonces, el campo de los ITS ha puesto a prueba muchas de las representaciones de conocimiento que han ido apareciendo, como las reglas, redes bayesianas o el razonamiento basado en casos, con mayor o menor éxito dependiendo de cada dominio particular.

En cualquier caso, tal y como nos cuenta Woolf (1992a), cuando el dominio que enseñamos se hace cada vez más amplio, la Inteligencia Artificial se convierte en el centro neurálgico que nos permite gestionar la ingente cantidad de información disponible. Es esta disciplina la que permite llevar más allá las capacidades de los simuladores y los sistemas multimedia permitien-

[32]Este sistema se denominaba SCHOLAR y enseñaba, mediante diálogo, geografía de Sudamérica.

do al sistema *razonar* sobre qué presentar en la siguiente interacción con el alumno y cómo. No se limita a proporcionar la respuesta correcta a las preguntas falladas. También responde al alumno en tiempo real, le enseña cómo resolver problemas, cómo probar (o desmentir) hipótesis, y le contesta a las preguntas (quizá ni siquiera previstas por los desarrolladores) de un modo similar a como haría un experto humano. Un ITS *comprende* lo que ocurre en la mente del estudiante, lo que le permite proporcionar ayuda contextualizada e individualizada (Siemer y Angelides, 1995).

Naturalmente, la comparación dada aquí entre los sistemas basados en marcos y los ITS es muy extremista. Como suele ocurrir en cualquier ámbito, el área de los sistemas educativos está plagado de tonos grises, existiendo un "continuo" entre lo que puede considerarse un sistema basado en marcos prototípico y lo que sería un ITS puro (Wenger, 1987). En cualquier caso la diferencia principal es lo que el experto transfiere al sistema: decisiones (enlaces entre marcos) o *experiencia* (conocimiento declarativo).

Por otro lado, esta visión tan optimista de los ITS puede resultar amenazadora para muchos profesionales de la enseñanza. En la práctica, sin embargo, la historia de la Inteligencia Artificial nos recuerda que el optimismo no es un buen consejero. Además, incluso en el muy lejano caso de que los ITS avanzaran lo suficiente como para poder ser utilizados como primera fuente de enseñanza, los profesores no podrían ser sustituidos. Tal y como ya se comentó en la sección 2.4, la *educación* tiene muchas dimensiones más allá de la mera *formación* como transmisión del saber humano. Los ITS "no son más que libros" sofisticados que se adaptan al nivel de conocimientos del alumno, le recuerdan cosas relevantes, profundizan en los aspectos no comprendidos o se saltan los que ya se saben. Pero libros, al fin y al cabo. Y éstos llevan siglos entre nosotros y no han desplazado al contacto humano y a la relación social proporcionada por un profesor de carne y hueso.

2.9.3. Conocimiento en los sistemas inteligentes de tutoría

En la sección anterior se mencionó que la diferencia principal es la inclusión de *conocimiento explícito* en los sistemas, en lugar de meras decisiones. Históricamente, éste es el conocido como *conocimiento del dominio*, es decir el relativo al área que el sistema intenta enseñar.

Existe un claro paralelismo entre el conocimiento del dominio y el poseído por un sistema experto sobre el mismo área. Resulta tentador, por ejemplo, asumir que podríamos utilizar *únicamente* el conocimiento integrado en MYCIN (Buchanan y Shortliffe, 1984) para *enseñar* a diagnosticar las mismas enfermedades infecciosas de la sangre que el sistema era capaz de identificar[33]. El verdadero problema estriba en que un ITS es un sistema *de*

[33]En realidad, se desarrolló un sistema de enseñanza sobre MYCIN conocido como GUIDON (Clancey, 1987), pero requirió añadir conocimiento adicional.

comunicación de conocimiento: no le basta saber sumar, debe ser capaz de *explicar* cómo se hace (y cómo no). Por tanto, no basta con que disponga de un *sistema experto* capaz de resolver las tareas a las que se enfrentará al alumno; también necesita tener la habilidad de conseguir que su "conocimiento" pueda ser adquirido por una tercera persona (el alumno)[34].

Para esto, necesita ser capaz de proporcionar explicaciones sobre cómo resolver los ejercicios, y sobre por qué las alternativas incorrectas proporcionadas por los alumnos lo son.

Aparte de las necesidades de comunicación que esto exige (en lenguaje natural, mediante diálogos, o de cualquier otra forma), esto último abre la necesidad de tomar decisiones sobre cuándo proporcionar ayuda y cómo. Esto requiere conocimiento sobre pedagogía en función del estilo de aprendizaje de cada estudiante (guiado, por autodescubrimiento, etcétera). Además es necesario decidir la forma en la que se *secuencia el contenido*, es decir qué conceptos se enseñan o ponen en práctica antes, y cuándo se da el salto a cuestiones más complicadas.

Esto último únicamente puede hacerse si el sistema es capaz de *estimar el conocimiento del alumno*. Es decir, para saber cuando ha llegado el momento de aumentar la complejidad de los ejercicios propuestos es necesario evaluar si el estudiante tiene la base necesaria.

Todo lo anterior demuestra que la cantidad de conocimiento que un ITS debe poseer es bastante variada. Eso ha llevado a considerar como organización clásica aquella que contiene cuatro tipos de conocimiento (Wenger, 1987; Woolf, 1992b):

- *Conocimiento del dominio*: contiene información sobre el área que el sistema enseña. Es el más importante, pues sin él no habría nada que enseñar, por lo que históricamente fue el primero que se incluyó (y, según del Soldato y du Boulay (1995) a veces el único que se ha seguido considerando).

- *Modelo del estudiante*: mantiene información específica sobre cada alumno. Al menos deberá seguir la pista sobre qué conceptos el alumno conoce ya, aunque una posible mejora sería registrar también *los errores* para tomar medidas que los solventen.

- *Conocimiento pedagógico*: contiene información sobre el proceso en sí mismo de enseñar. Basándose en el modelo del estudiante descrito antes y en este conocimiento pedagógico el sistema deberá ser capaz de decidir cuándo presentar conceptos nuevos y cuáles, o cuándo conviene repasar alguno ya descrito.

[34] Curiosamente, en el caso de GUIDON esto no resultó tan complicado. Al fin y al cabo, MYCIN ya era un sistema que servía para comunicar conocimiento: tras un diagnóstico, el sistema tenía que ser capaz de convencer a los médicos de que era correcto.

- *Conocimiento del interfaz*: decide las mejores formas de proporcionar las explicaciones necesarias al estudiante. Por ejemplo, podría incluir información sobre generación (y comprensión) de lenguaje natural, sobre organización de información en la pantalla o sobre la selección del mejor medio (texto, audio o vídeo) para comunicarse con el estudiante.

Generalmente estos cuatro tipos de conocimiento se han asociado a *módulos* en el lado de la arquitectura de la implementación de los ITS. De esa forma, cada módulo hará uso en ejecución del conocimiento *declarativo* asociado, dándole semántica. Por otro lado, de manera esporádica, algunos autores han considerado que el número de módulos debía ser mayor, apareciendo algún otro tipo de conocimiento adicional (Beck et al., 1996).

Las siguientes secciones describirán cada uno de los cuatro tipos de conocimientos clásicos con un poco más de detalle.

2.9.3.1. Conocimiento del dominio

Ya se ha mencionado que el conocimiento del dominio representa la materia que se está enseñando. Históricamente fue el primero en hacerse explícito, al abandonar la representación basada en marcos, iniciándose el camino hacia los ITS. Constituye, de hecho, su componente crítico (Inoue, 2000) pues encarna el conocimiento específico que fundamenta el sistema. Al fin y al cabo, para poder enseñar es necesario que el sistema adquiera, represente, almacene, acceda y muestre conocimiento (Kopec y Thompson, 1992, página 94).

El objetivo del conocimiento del dominio es doble:

- Debe ser *fuente* de conocimiento para proporcionar explicaciones, responder al usuario, o incluso generar preguntas para ponerle a prueba.
- Debe ser *generador de respuestas estándar* a las preguntas propuestas al alumno de modo que pueda compararse con la respuesta proporcionada por el usuario (Siemer y Angelides, 1995). Dependiendo del dominio, podríamos necesitar evaluar *todo el proceso*, no únicamente la respuesta final. En dominios complejos en los que existan varias soluciones (o "caminos" para llegar a ellas) esta capacidad de generación podría resultar bastante compleja.

El segundo uso asume una enseñanza basada en ejercicios que un posible "sistema experto" alimentado con el conocimiento del dominio sería capaz de resolver.

Si fuéramos capaces, también podríamos utilizar este conocimiento para *compararlo con el del alumno*, convirtiéndolo así en *objetivo de aprendizaje* en lugar de un mero "sistema experto" que proporciona explicaciones (Wenger,

1987). La dificultad de este tercer uso estriba, precisamente, en la necesidad de esa *comparación* entre el conocimiento del alumno y el del propio sistema para saber lo lejos que el usuario está de aprenderlo todo. Podríamos también encontrar los "errores de la reconstrucción" sufridos por el alumno durante su aprendizaje al rellenar (incorrectamente) los "agujeros de conocimiento", tal y como nos decía el constructivismo. Pero obviamente no podríamos quedarnos aquí, sino que tendríamos que proporcionar alguna solución a esos errores para cumplir la función dual de los ITS: facilitar la adquisición del conocimiento y proporcionar remedios a los errores del aprendizaje (Feifer, 1992).

No obstante, con el conocimiento del dominio siempre surge una limitación debido a la *granularidad*: en algún punto el sistema dará por comprendidos ciertos aspectos *que no será capaz de explicar*. Por ejemplo, en el caso de un sistema educativo sobre dinámica física, éste podría ser capaz de explicar la relación entre fuerza, masa y aceleración, pero ser incapaz de detallar la existente entre aceleración y velocidad por pertenecer al campo de la cinemática. Esto resulta una cuestión obvia cuando se ve desde el punto de vista del creador del sistema, pero resulta confuso por parte de los alumnos dado que un profesor humano raramente sufriría este tipo de lagunas.

El conocimiento del dominio debería también incluir cuestiones útiles para los aspectos pedagógicos que el sistema deberá mostrar. Así, será necesario enriquecer de algún modo este *conocimiento experto* para que sea consciente de qué conceptos son más complejos que otros, cuales son parecidos a otros para poder utilizarlos como ejemplo o para facilitar la abstracción por parte del alumno, etcétera. Algunos autores (como Kopec y Thompson, 1992) consideran esta información como *externa* al conocimiento del dominio, y la colocan en una categoría de conocimiento nueva a la que bautizan como *conocimiento del currículo*.

Por último, si pensamos en el *módulo* que dará utilidad a este conocimiento para, por ejemplo, resolver los ejercicios que él mismo propone, resulta de interés su *transparencia* en el sentido de que pueda analizarse su modo de proceder para generar explicaciones. Esto tiene relación con lo mencionado anteriormente sobre MYCIN y GUIDON y suele conocerse como *verosimilitud psicológica* ("psychological plausibility") en la que, a modo de ejemplo, los sistemas de razonamiento basado en casos estarían en un extremo y las redes neuronales en el opuesto.

2.9.3.2. Modelo del estudiante

La finalidad del modelo del estudiante es intentar conocer el nivel de habilidades y conocimientos del alumno (Kopec y Thompson, 1992) o, en general, del receptor de la información. Idealmente debería contener todos los aspectos del comportamiento y conocimiento del estudiante que puedan

ser tenidos en cuenta. Si comparamos en este sentido las posibilidades de los ITS y de los profesores humanos, los ordenadores quedan en clara desventaja debido a que la interacción posible es mucho más reducida (por ejemplo, un profesor puede apoyarse en la comunicación no verbal o en los tonos de voz). Afortunadamente, para desarrollar un ITS podemos conformarnos con un modelo parcial (Wenger, 1987). En cualquier caso, el modelo del estudiante tiene una gran importancia por su *naturaleza analítica*, que debe evaluar lo que el estudiante sabe a partir de sus respuestas.

Wenger (1987) asegura que la primera aparición de los modelos de estudiante en los ITS se la debemos a Fletcher (1975), aunque la idea de seguir la pista del alumno no era nueva. Los sistemas basados en marcos ya disponían de algo parecido para medir el rendimiento de los estudiantes en cada ejercicio. Sin embargo era mucho más sencillo (prácticamente era suficiente anotar por qué marco iba cada uno); en los ITS hay que mantener más información, a menudo estrechamente relacionada con el conocimiento del dominio. Esta colaboración entre ambos es la que genera la necesidad de transparencia en el *módulo experto* mencionada previamente.

La forma de modelar el conocimiento en el caso de los sistemas de marcos sufría una carencia importante: sólo anotaba lo que el usuario *sabía y no sabía*. Sin embargo, en la mayoría de las ocasiones un rendimiento *subóptimo* por parte del usuario al resolver un ejercicio no puede siempre achacarse a que aún no sabe los elementos del dominio, sino también a que hay cosas que *conoce mal*. El modelo del estudiante deberá también deducir esos *errores de reconstrucción* (*misconceptions* en inglés) para planear remedios específicos.

Al igual que ocurría en el caso del conocimiento del dominio, el modelo del estudiante se considera el componente declarativo principal del llamado *módulo* del estudiante, cuya finalidad es actualizar dicho modelo a partir de las acciones de éste. Para eso, debe analizar sus acciones y tratar de inferir las causas "profundas" que le llevan a darlas. Esto implica averiguar los conceptos que no ha comprendido bien a partir de sus errores, pero sin fiarse de los aciertos, que bien podrían ser producto de la casualidad o de una comprensión errónea pero que acierta en un caso particular[35]. Haciendo una analogía genética, Hollnagel denomina *genotipo* a la causa subyacente de un error y *fenotipo* a su manifestación observable (Hollnagel, 1991, 1993). A partir de ese *diagnóstico* se actualiza el modelo del estudiante con la nueva información recolectada.

A menudo se habla de modelo de estudiante a corto y a largo plazo (Rich, 1979, 1983). El primero es el utilizado durante la extracción de la causa subyacente de los errores cometidos en la sesión actual. El modelo a largo plazo mantiene información a un nivel de agregación mayor, agrupando las conclusiones que se han ido extrayendo del modelo a corto plazo, y que

[35]Por ejemplo, en el supuesto ITS para enseñar a sumar, esto ocurriría si se presentara una suma sin acarreo al alumno que no sabía "sumar llevando".

pueden ser útiles para la resolución de ambigüedades durante la extracción del *genotipo*.

En cualquier caso, para conseguir su objetivo, el modelo del estudiante debe guardar información de todo tipo (Siemer y Angelides, 1995):

- El conocimiento estimado del alumno.
- El conocimiento *erróneo* detectado y si ha demostrado ya o no haberlo superado.
- Una clasificación sobre sus conocimientos (principiante, medio, experto), típico de los modelos de usuario basados en estereotipos (Rich, 1979).
- Datos personales y estadísticos adicionales como el nombre, la velocidad de respuesta, si tiende o no a arriesgarse, los papeles que ha jugado y el rendimiento general en cada uno, etcétera.

Siemer y Angelides llaman a los modelos de estudiante con estereotipos *no basados en procesos*, en contraposición a los que sí lo son. Tratan de poner de manifiesto la gran ventaja de los modelos de estudiante *ejecutables*. Estos crean un "simulador" que es capaz de resolver los ejercicios que se le pondrán al estudiante *previendo* el resultado, lo que abre la puerta a la planificación (Wenger, 1987).

La labor de diagnóstico del módulo del estudiante es, al fin y al cabo, *generación de teorías* en el sentido de que a partir de *unos hechos* hay que proporcionar una teoría (un modelo del estudiante) que los expliquen. Conseguirlo no es en absoluto trivial, dado que las acciones del estudiante nos muestran únicamente la "punta del iceberg". Además, la cantidad de "ruido" es inmensa:

- Varios *errores en la reconstrucción* pueden aliarse para generar respuestas correctas.
- El modelo del usuario no deja de ser *una aproximación*.
- Los alumnos *no son consistentes*, es decir pueden tener suerte o equivocarse de manera fortuita por despistes.
- Los estudiantes pueden aprender de manera indirecta, externamente al propio sistema sin que éste sea consciente.

Esto hace que en ocasiones se hayan analizado las posibilidades de los *diagnósticos interactivos* como alternativa a los *diagnósticos inferenciales* que están detrás de lo expuesto anteriormente. Dado que averiguar a partir

de las pruebas (acciones) la razón que las ha provocado (modelo del estudiante) resulta muy complejo, podríamos de hecho "pedir ayuda" al estudiante preguntándole, por ejemplo, cosas concretas con la intención de desambiguar. Esto entraría dentro del área de la planificación interactiva.

Aunque hasta ahora se ha estado considerando que el modelo del estudiante se dedica a evaluar el desempeño del estudiante en el dominio, Mark y Greer (1993) proponen que se podría también realizar análisis de un mayor nivel de abstracción (más allá de la mera adquisición de conocimiento) para tener en cuenta cuestiones como capacidad de retención, tiempo de aprendizaje, porcentajes de completitud y *transferencia*, es decir la capacidad de aplicar los conocimientos aprendidos en un contexto a otras situaciones donde también pueden ser relevantes. Esto es especialmente importante cuando los alumnos aprenden en *entornos simulados* antes de entrar al mundo real.

2.9.3.3. Conocimiento pedagógico

Para ser un buen profesor no es suficiente con conocer en profundidad el área que se enseña, sino que también es necesario ser *un buen pedagogo*. Eso pasa por comprender el proceso mismo del aprendizaje y de la enseñanza.

Precisamente de este aspecto de la educación se encarga el conocimiento pedagógico, que contiene información sobre métodos y opciones para enseñar (Kopec y Thompson, 1992).

En los sistemas basados en marcos las *decisiones* pedagógicas se incrustaban en los enlaces entre los diferentes objetos de aprendizaje. El esquema clásico de los sistemas de tutoría inteligente opta por hacer explícito también ese *conocimiento pedagógico*, si bien, nos cuenta Wenger (1987) éste fue el último (de los cuatro) en llegar (y a menudo el gran olvidado).

La gran ventaja de tener la información pedagógica separada y especificada de manera declarativa consiste en que podría ser mejorado con el tiempo (sin afectar el resto de subsistemas), e incluso reutilizado entre diferentes sistemas. Desgraciadamente la independencia entre *el dominio enseñado* y el conocimiento pedagógico suele ser más una utopía que una realidad, por lo que la reutilización es increíblemente difícil de conseguir. A pesar de que para el observador profano la independencia del dominio parece ser una característica de la inteligencia, Freedman et al. (2000) aseguran que muchos expertos son, de hecho, muy escépticos en este punto. Así, es común opinar que la *habilidad pedagógica* depende en gran medida del dominio que se enseña. A modo de ejemplo, comenta que las analogías utilizadas al enseñar física (e incluso alguna de sus áreas concretas) son exclusivas de dicho campo y no pueden ser comparadas con otros dominios.

Como en los casos anteriores, el *conocimiento* pedagógico se encuentra embebido en el *módulo* pedagógico que tiene que tomar decisiones en dos niveles diferentes:

- A nivel global debe escoger lo que el usuario tendrá que realizar en el siguiente *episodio educativo* (*instructional episode*) o *escenario de aprendizaje*[36] (*learning scenario*). Éste podrá ser una explicación teórica, un ejercicio, o cualquier otro elemento intermedio.

- A nivel local debe tomar decisiones *durante* el episodio de aprendizaje actual. Esto sólo es posible si el seguimiento del estudiante es muy estrecho. En ese sentido se decide si interrumpir o no al alumno en su tarea cuando se detecta algún problema, y, si la decisión es afirmativa, qué decir.

En ambos casos se necesita, aparte del conocimiento pedagógico, conocer el *modelo del estudiante* visto en la sección anterior. En el caso de la elección del siguiente escenario de aprendizaje la parte de éste que nos interesa está enfocada a los conceptos ya conocidos, los desconocidos, y aquellos de los que se tiene una idea errónea. El conocimiento pedagógico necesario se referirá a suposiciones sobre cuándo es mejor repasar que introducir nueva materia, y al mejor modo de secuenciar los contenidos. Para esto último necesitaremos ayuda del conocimiento específico del dominio, que debería poder indicarnos qué partes tienen un nivel de dificultad mayor.

En lo referente al segundo tipo de decisiones, la información relevante del modelo del estudiante es aquella que describe las *preferencias en el aprendizaje*, es decir si un alumno particular prefiere libertad para experimentar o si prefiere un aprendizaje más guiado. A partir de esos datos, combinados con información pedagógica, se ha de decidir cuándo cada posible acción es deseable. Por ejemplo, no basta con decidir que en un determinado instante se interrumpirá al estudiante. También es necesario si nos limitaremos a pedirle que se replantee su último paso, si le daremos un ligero consejo para mejorar su rendimiento, una explicación completa sobre algún concepto que parece haber comprendido mal o, sencillamente, la solución. En cualquier caso, al realizar un ITS que *tiene la posibilidad de parar al usuario* se ha dado un salto desde la *ayuda reactiva* (proporcionada cuando el usuario la pide explícitamente o termina un ejercicio incorrectamente) hacia la *ayuda proactiva*, dado que el sistema decide tomar las riendas de manera autónoma cuando detecta que el estudiante podría necesitar su ayuda pero no la pide. Curiosamente esa ayuda proactiva podría aparecer incluso al *detectar inactividad*.

Conseguir que un sistema tome decisiones pedagógicas acertadas no es sencillo. Enseñar es un arte que requiere versatilidad. Por ejemplo, interrumpir al alumno siempre que se equivoca sin dejarle indagar por sí mismo puede romper la motivación e incluso estropear el aprendizaje. Pero no ayudarle nunca puede frustrarle.

[36] Según Rickel (1989), un *escenario de aprendizaje* es cualquier situación en la que tiene lugar "aprendizaje" por parte del alumno.

En este sentido, el propio Skinner tenía sus dudas, y así lo dejó reflejado en su cuaderno de notas que en 1980 publicara Epstein (Skinner, 1980). Cada pasaje venía precedido por un corto título, siendo uno de los más inquietantes el conocido como "¿Cuando ayuda el hecho de ayudar?":

> Al observar mi propia conducta con Lisa[37], lo que más me ha impresionado es lo siguiente: en mi afán por ayudar a la niña, destruyo las contingencias que le enseñarían a ayudarse a sí misma. Por ejemplo, aparto las ramas que le dan en la cara y la privo de la oportunidad de aprender a evitar las ramas. Le pongo los calcetines y la privo de la ocasión de que aprenda a hacerlo ella misma.

Si un influyente psicólogo tenía dudas sobre la educación de su propia hija, es obvio que los desarrolladores de ITS también tendrán graves problemas para "enseñarles" los momentos oportunos para interrumpir al estudiante.

En cualquier caso, la capacidad que tiene el sistema de intervenir en realidad depende mucho de la propia aplicación, porque requiere un "entorno de enseñanza" donde el usuario hace sus ejercicios o lee las explicaciones. Tenemos así un rango amplio de tipos de sistemas, desde los que el control lo tiene siempre el "tutor digital" y nada llega a descontrolarse (como ocurre en el programa que plantearemos en el capítulo 5), hasta el totalmente libre en el que el usuario puede hacer cualquier cosa que se le ocurra (como en el propuesto en el capítulo 6). En este último caso, el diagnóstico será mucho más complicado (debido a la explosión combinatoria de posibilidades), pero la experiencia puede ser muy educativa para el alumno. Se plantea aquí un conflicto semejante al que dentro del ámbito de la *ficción interactiva* es conocido como *dilema de la narración interactiva* (Berenguer, 1998) o *paradoja narrativa* (Aylett, 2000), donde es necesario buscar un equilibrio entre la libertad del jugador y el deseo del escritor por contar una historia.

2.9.3.4. Conocimiento del interfaz

Utilizando todo lo anterior, un ITS es capaz de realizar "decisiones profundas" sobre qué enseñar, cuándo interrumpir, y qué ayuda proporcionar. Pero necesita ser exteriorizado de algún modo; es necesario hacer llegar esa decisión al estudiante a través del *interfaz de usuario*. No podemos despreciar este aspecto, dado que es la *visión* que el estudiante tendrá del sistema completo. Un ITS puede tener un sofisticado diseño y unas impresionantes bases de conocimiento. Pero si el alumno *no comprende* lo que se le pretende decir, o comete errores por un complicado modelo de interacción con el sistema (y no por carencia de conocimiento) entonces el ITS no cumplirá su cometido (Mark y Greer, 1993).

[37] Lisa era su hija.

La necesidad de un interfaz de usuario, sin embargo, no necesariamente significa que el sistema necesite *conocimiento de interfaz*. En la organización clásica, no obstante, se menciona su existencia, seguramente debido a que cuando se propuso la mayor parte de las veces la interacción entre el sistema y el alumno *se realizaba en lenguaje natural*. Naturalmente en ese caso aparte de la *generación profunda* realizada por el resto de partes del sistema, se necesita la *generación superficial* que nos convierta la intención del sistema en discurso. Del mismo modo, es necesario el paso inverso, para entender lo que el alumno intenta preguntar o realizar. En ese caso, es clara la necesidad de este *conocimiento de interfaz*.

No obstante, ya en 1993 se reconocía que la situación había cambiado. En aquel momento, según Mark y Greer la mayor parte de los sistemas se basaban en gráficos y texto enlatado para mostrar información. Y como métodos de entrada usaban el ratón o, esporádicamente, el teclado. En este último caso no se usaba lenguaje natural, sino expresiones con una notación específica dependiente del dominio basada en algún tipo de lenguaje sencillo.

Aunque incluso en ese modo más simple de interacción podríamos continuar asegurando que tenemos "conocimiento" (por las plantillas de texto para la generación o los analizadores de entrada de lenguajes simples), hoy la mayor parte de las veces no se menciona explícitamente este conocimiento del interfaz. Como mucho, se reconoce la existencia de un *módulo* (concepto software) para el interfaz, pero no se le considera compuesto por conocimiento o por un modelo subyacente (Freedman et al., 2000).

2.10. Agentes pedagógicos

2.10.1. Introducción

Durante la década de 1.990 se pone de moda una nueva palabra: el término *agente software* aparece en infinidad de campos como algo novedoso y digno de mención[38]. No obstante, según Nwana (1996), los agentes software llevaban siendo un tema de investigación desde casi dos décadas atrás. Bradshaw (1997) sitúa el inicio del campo de los agentes (o al menos la fecha en la que se acuñó el nombre) todavía antes, poco después de la invención del término Inteligencia Artificial por John Mc Carthy a mediados de la década

[38]Los anglosajones denominan *buzzword* a palabras utilizadas con asiduidad en determinados ambientes que, sin embargo, tienen un significado que pocos conocen. Aunque la aparición original de esas palabras pueda tener, efectivamente, un fondo real y consensuado, su adopción por el público en general termina degenerando su uso que, muchas veces, sólo pretende impresionar a la audiencia para dar la sensación de conocimiento. Se incluyen así en frases con la intención de que sean difíciles de refutar debido al significado dudoso del término, pero por el que nadie se atreve a pedir explicaciones por vergüenza a dejar en evidencia su desconocimiento de la palabra del momento. Según la Wikipedia, algunos ejemplos son *framework*, *paradigma*, *3G* o *globalización*.

de 1.950.

Independientemente de estas discrepancias sobre los orígenes, se entiende por *agente software* a un sistema *autónomo* que es capaz de realizar tareas y lograr objetivos en entornos dinámicos y cambiantes (Muller, 1997). Para eso, han de observar el entorno, "comprenderlo", y actuar dentro de él para lograr sus objetivos, adaptándose a sus cambios. En ocasiones, los agentes pueden convivir con otros agentes para colaborar entre ellos con la intención de alcanzar los objetivos comunes. Un agente va más allá de la idea de *objeto* (de la programación orientada a objetos) porque no se define por sus métodos y atributos, sino por su *comportamiento*, que, además, es *autónomo* (lo que le permite ser *proactivo*). A menudo resulta difícil argumentar que un *programa* no es un agente. Franklin y Graesser (1996) lo intentan, para concluir que las características determinantes de un agente son reacción al entorno, autonomía, orientado a un objetivo y persistencia. Curiosamente, con esta definición se ven obligados a reconocer que un virus informático es, efectivamente, un agente.

También durante la década de 1.990 llegan al gran público los interfaces gráficos de usuario[39]. En ellos, existe una "metáfora" en el uso de un ordenador basada en un escritorio de trabajo tradicional. Son elementos comunes los botones y otros controles, los menús y los iconos, enriquecidos con operaciones como el conocido *arrastrar y soltar*. Una vez que este tipo de interfaces WIMP (*Windows, Icon, Mouse & Pointer*) deja de ser investigación, los visionarios comienzan a pronosticar lo que está por venir. Surgen así lo que se da por llamar *agentes de interfaz* (Maes, 1994). El problema principal esgrimido al tipo de interacción descrito estriba en que el ordenador continúa recibiendo órdenes sobre qué hacer: sigue siendo una *herramienta*. Los agentes de interfaz pretenden convertir a los ordenadores en *asistentes*, de modo que en lugar de proporcionar al sistema una serie de órdenes que ocasionen la realización de operaciones predecibles y bien definidas, lo que se le proporcionarán serán *objetivos* dejando el trabajo sucio de los detalles en el ordenador. En realidad, algunos agentes de interfaz se limitan a proporcionar ayuda sobre el propio sistema, mientras que otros buscan un cambio completo en la filosofía del propio interfaz.

Por otro lado, Nass et al. (1994) mostraron que la interacción con un

[39]Retrasar esta entrada hasta la década de 1.990 es en realidad un poco injusto para la firma Apple, que llevaba usando interfaces gráficos desde la aparición del Machintosh a principios de 1.984. Ellos fueron los primeros en crear ordenadores personales de éxito con este nuevo tipo de interacción, si bien la idea había nacido años atrás en Xerox PARC (*Palo Alto Research Center*, Centro de Investigación de Palo Alto). No obstante, la mayor parte del mercado de los ordenadores personales se fue hacia la arquitectura PC de IBM que ejecutaba MS-DOS con un interfaz basado en línea de comandos. He establecido la llegada masiva de los interfaces gráficos, por tanto, en la década de 1.990 por ser cuando apareció Windows 3.1, que supuso el verdadero pistoletazo de salida al éxito de la línea de productos Windows.

ordenador ocasiona *respuestas sociales* por parte de los usuarios. Es decir, la mayor parte de la gente *trata a los ordenadores como si fueran personas* incluso aunque el interfaz no nos incite a ello. Por tanto, una mejora sustancial a los agentes de usuario es aprovecharse de este hecho y *personificarlos* en "actores virtuales" que se muestran en el interfaz, y que reaccionan a las acciones del usuario. Es decir, la idea es proporcionar una representación visual del agente, que aumente aún más la visión antropomórfica de un ordenador que de manera natural parecemos tener.

Esto nos proporciona herramientas adicionales en la interacción, siendo la principal la *comunicación no verbal*. Los principios de diseño de este tipo de agentes son:

- Los agentes deberán obedecer a unas reglas sociales básicas.
- La representación visual deberá usarse para expresar las capacidades del agente subyacente.
- El personaje tendrá que mostrar una *personalidad* clara y consistente.
- No deberá distraer a los usuarios de su tarea principal, por lo que el control de la aplicación tendrá que estar, salvo en contadas y aceptadas ocasiones, en manos del usuario.

El agente de interfaz más conocido es el (a menudo odiado) clip de Office. Surgió por primera vez en Office 97, aunque Microsoft llevaba tiempo investigando en el uso de este tipo de agentes en el bautizado como *Proyecto Persona* (Ball et al., 1997). Su objetivo era crear asistentes informáticos que tuvieran una representación en pantalla y se comportaran como personas, tanto por la presencia visual como por el diálogo hablado. Con esto perseguían conseguir que los usuarios perdieran la probable reticencia a confiar en dichos agentes e interactuaran con ellos de un modo natural.

Dentro de la familia de productos Microsoft Home para el hogar, desarrollaron el fallido *Microsoft Bob*, que impulsaba un tipo de interacción completamente diferente, basada en el *interfaz de usuario social*. El objetivo era hacer a Windows 3.x más amigable, pensando especialmente en los usuarios más noveles. Para ello reemplazaban el administrador de programas, basado en la metáfora del escritorio propuesta por Xerox, con una pseudo-realidad virtual en la que los programas y documentos se presentaban de una manera más familiar y cercana a los objetos cotidianos. Por ejemplo, cuando se lanzaba el programa aparecía una puerta representando la "casa" del usuario. Un perro amarillo (llamado Rover Retriever o, simplemente, Bob) te da la bienvenida y te anima a llamar para entrar. Dentro se ve una habitación (figura 2.5) con las diferentes aplicaciones como objetos "familiares", como por ejemplo el calendario. Bob nos acompaña en nuestro vagar por este interfaz. Al hacer click sobre cualquier objeto que representa una aplicación,

Figura 2.5: Habitación principal de la casa de Microsoft Bob

ésta aparece con un interfaz del mismo estilo, apoyado con nuevos asistentes que dan un descanso a Bob.

En la práctica fue un tremendo fracaso, quizá por el aspecto infantil del sistema. Además los asistentes eran excesivamente cargantes, y los tutoriales se repetían una y otra vez independientemente de la facilidad del usuario para hacer las cosas. Como resultado, el uso del sistema terminaba siendo muy lento y tedioso, por lo que el proyecto fue abandonado[40].

El trabajo en Bob y el proyecto mencionado anteriormente terminó, no obstante, encontrando su fruto en los *ayudantes de Office* mencionados antes. Junto con él, surgió Microsoft Agent[41], un API para añadir fácilmente a los programas (para Windows) generación y reconocimiento de habla, así como la representación visual de un personaje que soporta varias animaciones del estilo de las del asistente de Office. Las aplicaciones pueden pedir al agente que ejecute esas animaciones que tienen un significado claro para el usuario, como alegría, confusión y tristeza, y algunas otras de aplicación en ciertos contextos, como señalar en distintas direcciones, escribir, leer, llamar la atención, etc. Naturalmente, es responsabilidad de la aplicación utilizar esas animaciones en momentos adecuados para crear la sensación de "inteligencia" del personaje.

Pero dejemos de lado, por el momento, a los agentes de interfaz. En la

[40] Como curiosidad histórica, según la Wikipedia Microsoft terminó intercambiando el dominio `bob.com` con Bob Kerstein por el de `windows2000.com`

[41] `http://www.microsoft.com/msagent/`

sección anterior mencionábamos que los ITS añaden complejidad a los sistemas basados en marcos en lo que se refiere a la resolución de ejercicios. De sencillos tests se llega a entornos de aprendizaje en los que el alumno se enfrenta a ejercicios que requieren una interacción más compleja. Como hemos visto, inicialmente se utilizaba lenguaje natural (quizá intentando emular el método socrático), y luego se aprecia un desplazamiento a entornos gráficos donde el *ratón* cobra especial importancia. Esto simplifica la implementación al desaparecer el conocimiento del interfaz.

En cualquier caso, resulta habitual que el sistema proporcione un *entorno* en el que el alumno resuelve los ejercicios. En función del tipo de ITS, éste podría limitarse a esperar a que el alumno termine completamente y realizar un análisis *a posteriori* del resultado. En este sentido el ITS se asemejaría en su funcionamiento a un sistema basado en marcos, salvo por la complejidad adicional de la fase de corrección de la solución del estudiante.

La intención de la mayor parte de los ITS, sin embargo, es proporcionar una ayuda más cercana, monitorizando continuamente al estudiante, y decidiendo dinámicamente qué ayuda proporcionarle, ya sea de manera reactiva o proactiva.

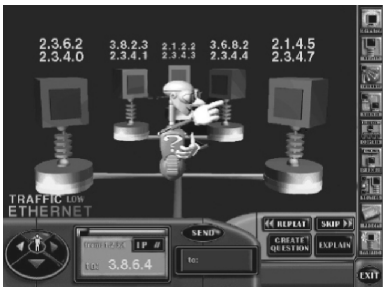
En la década de los 90 el *hardware* de los ordenadores personales consigue unos rendimientos lo suficientemente altos como para que la representación en tiempo real de contenido gráfico comience a dejar de ser un problema. Esto acercó el mundo de los simuladores al gran público. Además, permitió que aparecieran sistemas de enseñanza con un gran contenido visual, más allá de imágenes estáticas o vídeos no interactivos. No tardaron en surgir sistemas en los que el entorno de aprendizaje (donde se resolvían los ejercicios) eran bidimensionales o tridimensionales e interactivos, dando un paso más a los habituales interfaces basados en menús (Trindade et al., 1999; Dede et al., 1996).

Si queremos convertir estos *entornos virtuales* en verdaderos *sistemas educativos* es necesario añadirles la capacidad de *enseñar*, ayudando al estudiante cuando éste lo reclama, proporcionando información sobre lo bien o mal que lo está haciendo, y proponiendo ejercicios acordes al nivel del usuario.

La posibilidad de añadir las técnicas utilizadas en los ITS a los entornos virtuales aparece pronto en la comunidad. Inicialmente utilizaban el tipo de interacción en el que se proporciona *feedback* al estudiante *tras* cada episodio de aprendizaje.

Proporcionar información *durante* la resolución de un ejercicio resulta en realidad un poco más complicado. En los sistemas con interacciones basadas en ratón se podía mostrar una ventana, o reservar una pequeña parte de la pantalla para el texto explicando aquello que el sistema decide hacer notar. Sin embargo en un entorno *inmersivo* (independientemente de que sea en 2D o en 3D) esto resulta bastante más artificial.

(a) Herman the bug (Stone y Lester, 1996)



(b) Cosmo (Lester et al., 1997b)

(c) Steve (Rickel y Johnson, 1999)

(d) Adele (Shaw et al., 1999)

Figura 2.6: Ejemplos de agentes pedagógicos

Aquí es donde nuestro recorrido por los agentes de interfaz cobra sentido. Bajo este contexto surgen los *agentes pedagógicos* que mezclan ideas del mundo de los ITS de monitorización constante, de los agentes software y de los agentes de interfaz. Se convierten así en elementos software autónomos, pero especializándose en una tarea concreta: ayudar al estudiante durante su aprendizaje en un entorno interactivo. En ese sentido, el agente software "reside" en el entorno de aprendizaje, recibiendo como "estímulos" las acciones del usuario, y realizando operaciones en ese entorno que resultan ser intervenciones para ayudar al alumno. Además, disponen de una *representación visual* que les proporciona un acercamiento al usuario, les permite proporcionar comunicación no verbal e incluso abre la puerta a que el propio agente *interactúe con el interfaz del entorno de aprendizaje* de una manera más natural y menos intrusiva para el usuario. La figura 2.6 muestra cuatro conocidos agentes pedagógicos.

2.10.2. Características de los Agentes Pedagógicos

Los agentes pedagógicos son, por lo tanto, agentes especializados que habitan los entornos interactivos de aprendizaje para potenciar ese apren-

dizaje, y que disponen una representación visual. Debido a esto, a menudo son llamados *agentes pedagógicos animados*. Podemos pensar en ellos como la *encarnación* de la parte de un ITS que ayuda a lo largo de cada episodio de aprendizaje. Naturalmente, igual que éstos, se adaptan al estudiante evaluando sus conocimientos para ajustar su interacción tal y como haría un profesor. Pero además, debido a su representación física, puede incitar al alumno a interactuar con él preguntándole, o mostrar reacciones en cierto modo *humorísticas* que hagan más amena la interacción con el sistema. Conseguir que el alumno "vea" al agente que le está enseñando a través de una figura en movimiento que crea la ilusión de tener vida (life-like) tiene a menudo repercusiones positivas en la motivación. En general, se cree que los agentes pedagógicos animados capturan la imaginación de los estudiantes, lo que les hace sentirse atraídos por el entorno de aprendizaje (Lester et al., 1997a,b).

Los agentes pedagógicos heredan todas las dificultades de implementación tanto de los agentes como del software educativo. El uso de una representación animada para mostrarlo supone problemas nuevos. En primer lugar, los agentes deben mostrar un comportamiento coherente, coordinándolo con el de otros agentes y respondiendo de forma lógica a los estímulos de su entorno, incluyendo dentro de éstos a las acciones del usuario. Además, necesitan poseer el conocimiento sobre el dominio que el estudiante está aprendiendo. En general los agentes comunes tienen cierto grado de inteligencia que les permite desenvolverse en su entorno para conseguir sus objetivos. En el caso de los agentes pedagógicos (igual que en los ITS) esa inteligencia no consiste sólo en poder resolver los problemas a los que se enfrentan (es decir, los ejercicios que deben solucionar los estudiantes), sino ser capaz de explicar cómo se resuelven, dando consejos y ayuda contextualizada. Esto requiere una profunda comprensión de las relaciones entre cada una de las acciones necesarias para solucionar el problema.

Si el agente pedagógico es animado, hay que conseguir una armonía entre sus explicaciones y su representación para hacerlo "creíble". Con esto no sólo nos referimos a conseguir una sincronización entre el movimiento de la boca y las palabras dichas cuando el agente habla, sino a producir un comportamiento natural y apropiado para el papel que juega dentro del entorno virtual.

Como ya se ha dicho, una ventaja adicional de los agentes pedagógicos animados es que pueden hacer uso de comunicación no verbal. Mediante gestos de alegría, duda, o impaciencia, pueden rápidamente y sin necesidad de hablar indicar al alumno que está haciendo las cosas bien, mal, o que debería plantearse realizar alguna acción urgentemente. Naturalmente, hay que tener cuidado durante el diseño con el uso de estos movimientos para que el alumno no se distraiga. Además, en ocasiones esos movimientos pueden llegar a causar que desaparezca la sensación que el usuario tiene de que el agente

es un ente vivo. Por ejemplo, si siempre que el alumno hace algo bien el agente ejecuta el mismo gesto (digamos asentir), el usuario podría terminar considerándolo una respuesta automatizada y carente de todo sentimiento. Para evitarlo, los diseñadores suelen meter diferentes animaciones para indicar una misma cosa, eligiendo una u otra de forma más o menos aleatoria. Esa variedad hace que la ilusión de vida se conserve, al menos, una mayor cantidad de tiempo.

Los gestos que el agente realiza tienen también un importante papel para la motivación del estudiante, pues muestran respuestas emotivas del agente. El agente puede felicitar al usuario cuando logra resolver un ejercicio especialmente difícil, y acompañarlo con un gesto de alegría. Si el usuario se confunde reiteradamente, su respuesta será la contraria, mostrando una expresión de pena. Todo esto entra en realidad en el campo del razonamiento afectivo. Según Elliott et al. (1999), conseguir agentes que muestren emociones a los alumnos tienen varios beneficios para el aprendizaje:

- Dan la impresión de que el agente se preocupa por el progreso del estudiante. Esto hace que el alumno sienta que el agente "está con él" en su tarea de aprender, lo que le anima a preocuparse por su propio progreso y la opinión que el agente tiene de él.
- Si el agente es "consciente" de las emociones del estudiante puede animarle y ayudarle cuando detecta su frustración o pérdida de interés.
- Si se consigue que el agente muestre entusiasmo por la materia que está enseñando, el estudiante podría adquirir dicho entusiasmo y encontrar más atrayente el aprendizaje.
- Los agentes con rica e interesante personalidad pueden hacer que el aprendizaje sea más divertido, pues el estudiante puede disfrutar su interacción con él, lo que creará una percepción más positiva del entorno de aprendizaje. De este modo, el alumno tenderá a utilizar más el software educativo, con los claros beneficios que esto supone.

La fuerte presencia visual mostrada por los agentes animados puede, por lo tanto, generar un *impacto afectivo* en los usuarios que potencie el aprendizaje gracias a lo que Lester et al. (1997a) llaman "el efecto persona". Comparando la reacción de un grupo de alumnos ante el uso del mismo software educativo en el que lo único que cambia son las características del agente pedagógico, llegan a la conclusión de que la inclusión de tales figuras mejora la relación entre el alumno y el sistema. Aunque reconocen que su estudio no es muy completo y deberían llevarse a cabo investigaciones más detalladas, aconsejan a los creadores de entornos interactivos de aprendizaje que incluyan agentes pedagógicos animados, incluso aunque éstos no proporcionen consejos.

Este resultado es un poco controvertido y ha sido revisado varias veces. Por ejemplo, Mulken et al. (1998) llega a la conclusión de que, efectivamente, los agentes pedagógicos son beneficiosos en las *medidas subjetivas* como el entretenimiento y la dificultad percibida del aprendizaje, pero no ocasiona ningún efecto significativo en las *medidas objetivas* como la comprensión y la memorización. Algo parecido concluyen Moreno et al. (2000)[42], añadiendo que lo realmente importante para que el alumno recuerde lo que se le está enseñando es que el sistema lea las explicaciones, en lugar de tenerlas que leer él mismo. De ahí deducen que los creadores de software educativo deberían prestar especial atención a la generación del lenguaje hablado, en lugar de a la representación de personajes animados. Su estudio, sin embargo, se centra en el aprendizaje, es decir en la cantidad de información que los alumnos pueden recordar tras utilizar el sistema, pero no en la impresión que les ha causado su uso. En ese sentido, Moundridou y Virvou (2002) vuelven a confirmar que los agentes pedagógicos hacen considerablemente más amigable la interacción entre el usuario y el sistema, y los estudiantes suelen considerar interesantes los programas de enseñanza que poseen uno de tales agentes.

Por último, Prendinger et al. (2003) vuelven a evaluar la veracidad de todos esos resultados, pero esta vez de un modo más objetivo: haciendo uso de información fisiológica para medir las respuestas autónomas del sistema nervioso de los usuarios asociadas a diferentes sucesos en el interfaz. Los resultados muestran que, efectivamente, un personaje animado disminuye el *estrés del usuario* ante los ejercicios y tiene un efecto positivo en la dificultad aparente de éstos.

Lo que parece que nadie se ha planteado es la posibilidad de que los resultados de los experimentos que aseguran esto último se vean en realidad falseados por culpa de la "novedad". Actualmente, los agentes pedagógicos se consideran muy interesantes, y los alumnos los suelen puntuar positivamente. Se concluye por lo tanto que la educación por ordenador es buena, y motivante. Lo que no contestan es si lo es en general, o solamente hoy en día. Por ejemplo, la historia de los programas de televisión o, mejor aún, de los juegos de ordenador, muestra que muchos de los que tuvieron gran éxito son vistos actualmente como muy simples, sencillos de superar e incluso aburridos. Da la impresión de que la sociedad los asimila, y pierde el interés por ellos. Naturalmente, la creación de nuevos juegos con mejores características técnicas y lúdicas tiene buena culpa de ello. Sin embargo no sólo es cuestión de pensar en juegos o programas concretos, si no en el concepto general de ordenador o televisión. Cuando aparecieron por primera vez, la sociedad se revolucionó. Hoy en día no son más que electrodomésticos comunes[43]. Hoy

[42] Curiosamente, uno de los coautores de este artículo lo era también del original que acuñó el término "efecto persona".

[43] Alan Kay, pionero de la orientación a objetos y los interfaces WIMP, resume la silenciosa aceptación de lo que un día fue novedoso diciendo que "la tecnología es aquello que aún no existía cuando nací".

una figura animada dentro de un ordenador que muestra sentimientos es algo nuevo que atrae a los usuarios. Tal vez en unos años se vuelva algo habitual y carente de interés.

Por otro lado los experimentos que tratan de probar los beneficios de los programas educativos a menudo se realizan con grupos de control en colegios. Los alumnos son sacados de la clase con su profesor para hacer uso de ordenadores. Y salir de la rutina diaria es algo que todo el mundo considera interesante, aunque sea para recibir aburridas clases de educación vial. Si dentro de muchos años se hace común la educación por ordenador, quizá lo realmente motivante sean las clases con profesor, por ser lo que se sale de la rutina.

En cualquier caso, lo que nos incumbe es construir sistemas que sean interesantes y útiles, aunque sólo lo sean durante unos años. Cuando se presenta a los estudiantes un tema por primera vez, suele ser necesario demostrarles cómo resolver problemas relacionados, y el modo de ejecutar ciertas tareas. Eso ha originado que a algunos agentes pedagógicos (Johnson et al., 1998; Rickel y Johnson, 1999) se les haya incluido la capacidad de poder resolver los problemas que el usuario debe superar. De ese modo, los alumnos pueden también aprender a través de ejemplos. Naturalmente, para que esto valga la pena el estudiante deberá prestar atención a los pasos que el agente pedagógico da, aunque éste no es un problema nuevo y lo han sufrido los profesores durante siglos. Las demostraciones por ordenador de cómo resolver ejercicios se han realizado habitualmente mediante presentaciones multimedia. Aunque en los sistemas con agentes pedagógicos se podría continuar utilizando esta técnica, en general se prefiere que sea el propio agente el que resuelva el ejercicio porque la demostración puede adaptarse a diferentes estados del entorno, o a distintos estados iniciales. Además, si el entorno de aprendizaje es un entorno tridimensional, el usuario podrá ver la presentación desde cualquier punto, y el agente adaptará sus gestos en función de la posición del usuario. Por último, la capacidad de ejecutar tareas de los agentes pedagógicos puede utilizarse como caso extremo cuando el alumno no sabe continuar a mitad de un ejercicio. El agente podría continuar el trabajo, mostrando cómo realizar sólo partes del escenario completo en lugar de todo el ejercicio como ocurría con las demostraciones multimedia.

Aunque la posibilidad de que el agente resuelva el problema es interesante como último recurso, en general es preferible que antes de llegar a ese extremo pueda proporcionar ayuda con diferentes niveles de detalle. El principal objetivo de los agentes pedagógicos animados es guiar a los estudiantes a través de materias complicadas y, sobre todo, ofrecerles consejos adaptados al contexto del episodio de aprendizaje actual. Esos consejos no deben ser detallados desde el principio, pues hacen que el alumno no necesite razonar cuando pregunta al agente. Es más interesante que en primera instancia el estudiante reciba una respuesta vaga, pero correcta, que le inste a tratar

de encontrar la solución por sí mismo. Si aun así, el usuario sigue perdido, puede volver a pedir ayuda, a lo que el agente irá dando información cada vez más concreta. De ese modo, el alumno puede parar de preguntar cuando considere que ya ha entendido su problema, y, si es incapaz de solucionarlo por sí mismo, en última instancia el agente le puede dar todos los detalles o, incluso, solucionarlo él mismo.

Por otro lado, según Stone y Lester (1996) los alumnos encuentran muy molesto que el agente pedagógico repita la misma explicación, o dé los mismos consejos. Para evitarlo, es interesante tener al menos dos versiones para la misma explicación, una detallada y otra del estilo de "Recuerda que...", lo que incrementa la ilusión de inteligencia del agente. Naturalmente el sistema tendrá que controlar qué explicaciones se han dado y cuándo, para no repetirlas a no ser que haya pasado el tiempo suficiente.

Una ventaja adicional de tener varios niveles de ayuda o las explicaciones abreviadas es que pueden ser utilizadas por el agente sin que el usuario lo solicite. Si el usuario muestra dificultades con el ejercicio actual, ya sea porque se equivoca o porque se siente incapaz de realizar ninguna acción y se queda parado, el agente puede dar consejos no solicitados para tratar de ayudar al alumno a superar su desagradable situación sin necesidad de preguntar. Naturalmente los diseñadores deben vigilar estas intervenciones automáticas pues en ocasiones pueden resultar innecesarias y molestar al usuario. Eso muestra que hay una delgada línea entre un agente pedagógico útil y uno que es más una distracción que una ayuda. De hecho, los agentes pedagógicos resultan interesantes en el software cuando éste es educativo y el sistema es utilizado para enseñar. La inclusión de agentes pedagógicos en sistemas que son utilizados por expertos suele ser más una molestia que una ayuda, pues el experto no necesitará prácticamente nunca los consejos de un agente que posiblemente "sepa" menos que él.

Los agentes pedagógicos heredan las ventajas en lo que se refiere a enseñanza individualizada que ya proporcionaban los ITS. En cierto modo, un sistema educativo que posea uno de estos agentes puede seguir considerándose un ITS pero con un *interfaz más sofisticado*. Esto supone un esfuerzo de implementación considerable, que constituye un área de investigación por sí misma: la creación de agentes virtuales que habitan en entornos 3D y proporcionan comportamientos creíbles. Es necesario conseguir crear la ilusión de que el agente es consciente de su entorno, apuntando a los objetos de los que habla, "mirando" al estudiante durante una conversación con iniciativa mixta, o llamando la atención cuando el estudiante parece desviar su atención mientras se le habla. Desde nuestro punto de vista, esto hace reaparecer el *conocimiento del interfaz* que, en cierto modo, desapareció de los ITS con el abandono de la interacción con lenguaje natural.

Sin embargo, en algunas circunstancias esto supone una dificultad adicional. En entornos de enseñanza tridimensionales, el agente pedagógico deberá

ser consciente de la localización de cada elemento para interactuar con ellos de un modo creíble cuando esté mostrando como resolver un determinado ejercicio. La información necesaria para esto acabamos de colocarla como *conocimiento de interfaz*, al igual que al resto de conocimiento del agente pedagógico. Sin embargo, en ocasiones el propio entorno *es el objetivo de aprendizaje*. Si se está simulando, por ejemplo, el interior de la sala de máquinas de un barco o una central nuclear para enseñar a manejarlas, el alumno también tendrá que *aprender la estructura de dicho entorno*. Esto supone un problema en la implementación, dado que lo que el agente pedagógico necesita para desenvolverse en el entorno se convierte a la vez en *conocimiento del dominio* que hay que enseñar. El sistema no sólo debe conocer su entorno y usarlo, sino que también *debe explicarlo*, ocasionando complejidades adicionales durante la implementación.

Conclusiones

Iniciábamos el capítulo hablando de la gran importancia que la educación tiene en la civilización. Las ventajas que ésta proporciona a los ciudadanos (a nivel individual y global de la sociedad en la que habitan) han sido descritas en la sección 2.2.

Sin embargo, la sección 2.3 nos ha mostrado que la educación *es muy cara*. Eso ha llevado a que durante muchos siglos el hombre haya analizado la forma en la que aprende, con la intención de *optimizar la forma en la que enseña*. En el apartado 2.4 se han analizado las dos tendencias más influyentes, el conductismo y el cognitivismo, para hablar después, en el 2.5 del propio *sistema educativo* que se supone ha de aplicarlas.

Lo anterior sirve como base que permite entender las diferentes aproximaciones seguidas por las distintas familias de programas informáticos educativos. Hablábamos de los sueños de Thorndike que se vieron por fin desarrollados en los sistemas basados en marcos descritos en la sección 2.7, y de los simuladores que ahorran tanto vidas humanas como ingentes cantidades de dinero en la sección 2.8.

Los simuladores en realidad siguen las teorías cognitivistas del aprendizaje, también seguidas por los sistemas inteligentes de tutoría descritos en la sección 2.9. Éstos beben de las ideas de la Inteligencia Artificial para dar un paso más allá tanto a nivel de implementación como de características de tutoría. La investigación en este campo ha ido de la mano de las diferentes técnicas de representación del conocimiento, necesario en varias partes del sistema.

Los últimos en llegar a los programas educativos han sido los *agentes pedagógicos* de la sección 2.10. Surgen como *personajes* representados en *entornos de aprendizaje*, mejorando la sensación (subjetiva) que los usuarios tienen hacia el propio sistema, pero que acrecienta las dificultades de imple-

mentación al agrupar los requisitos de los ITS y de los agentes virtuales.

En realidad, los simuladores son considerados herramientas *de apoyo* al aprendizaje porque es normalmente necesario algún tipo de ayuda externa a su alrededor. En concreto, en principio un simulador no escoge por sí mismo los ejercicios a los que enfrentar al alumno, ni realiza ningún tipo de análisis sobre el modo en el que éste ha reaccionado ante él. Por otro lado, los agentes pedagógicos pueden ser considerados como mero *interfaz* que apoya al resto del sistema. Naturalmente, esta visión es bastante simplificada y no deberían ser despreciados: su implementación puede resultar muy compleja y sus beneficios en el resultado final son significativos.

En cualquier caso, esto deja por tanto únicamente dos grandes tipos de sistemas: los basados en marcos y los ITS. Los primeros son conductistas, algo bueno sólo en algunos dominios. Su implementación requiere mucho tiempo en la creación de *contenido*. Los ITS intentan suplir esto haciendo que el contenido sea "generado" automáticamente a partir de *conocimiento explícito* añadido en el sistema. Esto permite que la implementación escale bien a dominios amplios y a aproximaciones cognitivistas. El coste a pagar es, naturalmente, la creación de ese conocimiento, cuya dificultad varía dramáticamente con el dominio a enseñar y encauza con las más horribles pesadillas de la Inteligencia Artificial. En el capítulo 4 describiremos la solución que proponemos a este desplazamiento del trabajo de autoría desde el contenido al conocimiento explícito.

En el próximo capítulo...

Los agentes pedagógicos tratan de mejorar la interacción hombre-máquina y buscan la motivación, cuya repercusión en las medidas objetivas sobre el aprendizaje resultan aún algo discutibles, pero que mejoran sustancialmente la sensación subjetiva sobre el entorno de aprendizaje.

Un modo todavía mayor de conseguir interés por el uso del *software* educativo es mezclarlo con el *entretenimiento* proporcionado por los videojuegos. Sus usuarios tienen suficiente motivación como para dedicarles largas horas a pesar de su, a menudo, increíble dificultad y necesidad de repetición. El próximo capítulo se adentra en este mundo y en las alternativas que ya existen.

Capítulo 3

Videojuegos y videojuegos educativos

La madurez del hombre es haberse reencontrado, de grande, con la seriedad que de niño tenía al jugar.

Friedrich Nietzsche

Resumen: En este capítulo se describe la importancia del juego para el aprendizaje, y las repercusiones que está teniendo el uso de videojuegos entre los jóvenes de hoy. Este nuevo modo de ocio ha llegado incluso a ocasionar un cambio en los intereses y habilidades de sus usuarios, provocando un abismo generacional sin precedentes. El uso de *videojuegos educativos* es analizado como una alternativa válida para solventar los problemas de una educación que insiste en mantener un modelo creado hace demasiado tiempo en un mundo que no era, ni de lejos, como el de hoy.

3.1. Introducción

Existe la opinión generalizada de que el nivel educativo sufre de unos años a esta parte una pendiente negativa muy pronunciada. Problemas de interés, falta de atención, y la política del no esfuerzo son el día a día de los docentes de muchos países desarrollados.

El fracaso de los alumnos es en realidad una herida abierta que prueba el fracaso de los profesores. Para aliviar conciencias, la solución es hacer decrementar la carga de contenidos en los planes de estudio, con la esperanza

de unos falseados mejores resultados que permitan dormir tranquilos a los máximos responsables.

Sin embargo, tampoco esta alternativa está demostrando ser la mejor solución. Si la causa de todos los males son los tutores y/o los profesores (usar a los alumnos como chivo expiatorio vuelve a ser un modo de tranquilizar conciencias) es sólo importante en parte. Lo que verdaderamente es *imprescindible* es buscar soluciones.

Sherlock Holmes solía decir que cuando todas las demás hipótesis han demostrado ser falsas, la última, por muy inverosímil que resulte, será la verdadera. Quizá esté la solución en la alternativa, también para muchos inverosímil, presentada en este capítulo sobre videojuegos.

3.2. Los juegos como medio para el aprendizaje

Jugar es, según la RAE (Real Academia Española), "hacer algo con alegría y con el solo fin de entretenerse o divertirse". Aunque esta definición irradie un carácter lúdico, es universalmente reconocido la estrecha relación entre juego y aprendizaje. Los cachorros de león juguetean y se divierten, sin saber que, inconscientemente, están preparándose para una vida adulta de cacería gracias al ensayo y perfeccionamiento conseguido mediante el juego. El impulso de *imitar* lo que ven en los miembros más mayores de la manada les resultará increíblemente útil en el futuro.

Los bebés juegan de *motus propio*, instinto que les lleva a moverse, gatear, andar o manipular objetos. Piaget denomina a estos juegos prematuros *juegos motores*, que les incitan a tener sus primeras interacciones con el mundo que les rodea, descubrir su propio cuerpo y cómo controlarlo. Los adultos hemos olvidado la importancia que tuvo en nuestro crecimiento el golpear reiteradamente diferentes objetos contra el suelo, pero consideramos que, *de manera innata* somos capaces de prever qué ocurrirá si algo cae desde una cierta altura. Afortunadamente, aunque muchos padres y cuidadores insistan en impedir en los niños este tipo de "extraños" comportamientos, los bebés los repiten una y otra vez hasta que consolidan los resultados y pierden el interés.

Cuando estos primeros titubeos por la vida comienzan a realizarse en grupo, el juego pasa a ser una herramienta primordial para el desarrollo *psicosocial*. Los niños comienzan a relacionarse, a poner a prueba sus habilidades y conocimientos, así como a desarrollarlos gracias al acercamiento a sus semejantes. La aparición de los *juegos con reglas* suponen un nuevo reto, ya que hay que *aprenderlas y no romperlas*. No es posible hacer cualquier cosa, sino que hay que seguir unas normas preestablecidas que normalmente están en desacuerdo con los propios intereses. Eso despierta el razonamiento, la competencia, o, según el caso, la cooperación y la negociación. No es de extrañar, por lo tanto, la opinión de Rieber (1996), para quien el juego

tiene, especialmente durante la infancia, un papel primordial en el desarrollo psicológico, social e intelectual.

Lo que más nos interesará posteriormente de todo esto es que, a pesar del *esfuerzo* por aprender esas reglas, el juego es una actividad *motivante* que surge del propio individuo[1]. Además, como se ha dicho, la importancia que tiene en la educación es inmensa, pues aumenta las habilidades motrices y verbales, aviva el ingenio, despierta la capacidad de observación y fomenta la paciencia.

Todo esto convierte al juego en algo *esencial* para la adaptación social y familiar, y, naturalmente, para el crecimiento mental. La importancia es tal que el juego es considerado *un derecho* por la Convención sobre los Derechos del Niño, que dice, en su artículo 31:

> Los Estados Partes reconocen el derecho del niño al descanso y el esparcimiento, al juego y a las actividades recreativas propias de su edad y a participar libremente en la vida cultural y en las artes.

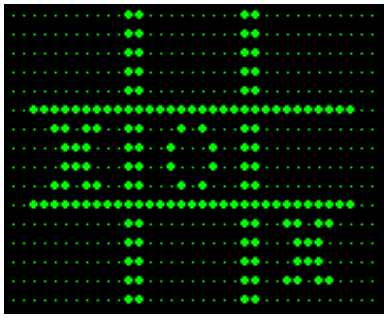
Las oportunidades de aprendizaje que proporcionan los juegos y juguetes son bien conocidas por los propios fabricantes, que tienen diferentes alternativas dentro de la familia de los juegos educativos. Juegos de construcción (lego, mecanos), de inteligencia (ajedrez, damas, Rummykub, Conecta 4), de memoria (Memory, Simón), etcétera son conocidos por todos. En este sentido, los padres preocupados por los juegos que les regalan a sus hijos a menudo se preguntan cuales son más adecuados para su edad, para maximizar el desarrollo del niño[2].

Todo esto demuestra que en general la sociedad acepta la importancia del juego para el desarrollo infantil. Sin embargo, también hace que, de manera inconsciente, muchas veces se considere al juego como *cosa de niños*. Por desgracia durante la vida adulta normalmente tendemos a autoimponernos la obligación de justificar la mayoría de las acciones que realizamos buscándolas una *utilidad*. Al considerar el juego como algo *voluntario*, realizado *por placer* y, especialmente, *ausente de finalidad*, resulta complicado justificarlo desde el punto de vista práctico.

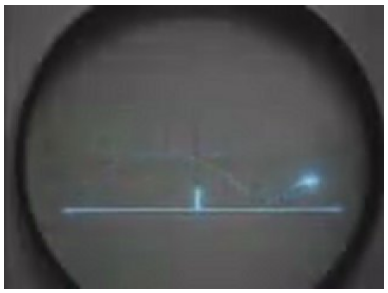
Resulta triste esta "seriedad" de la vida adulta, que a menudo nos roba la posibilidad de seguir jugando. Se dice que uno no deja de jugar al hacerse mayor, sino que se hace mayor al dejar de jugar. De hecho, es claro que el placer por el juego no desaparece, tal y como mostraría cualquier visita

[1] Después de todo, si alguien es *forzado* a jugar, la actividad dejará, en cierto modo, de ser un juego.

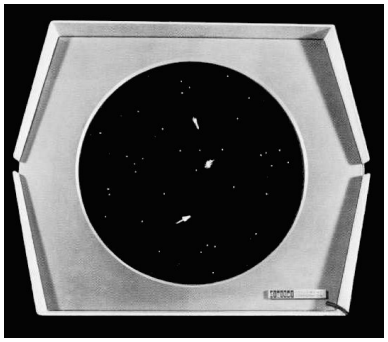
[2] Un sitio Web interesante a este respecto es `http://www.ludomecum.com/`, creado para facilitar y orientar en la búsqueda de juegos y juguetes educativos. Fue fundado, entre otros, por el Dr. Gonzalo Jover, catedrático de Teoría de la Educación de la Universidad Complutense de Madrid, y dispone de información que categoriza más de 700 productos.



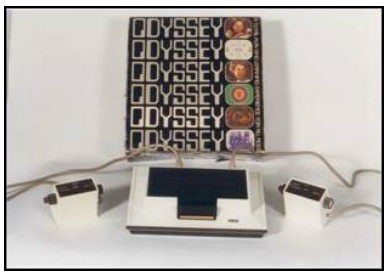
(a) Tres en raya en el EDSAC (1.952)



(b) Tennis for Two (1.958)



(c) Space War (1.961)



(d) Odyssey (1.972)

Figura 3.1: Primeros videojuegos y juegos de ordenador

a los centros de la Tercera Edad. Pero la carencia de tiempo obliga a la optimización, que se encarga de eliminar las dedicaciones "superfluas" de nuestras ajetreadas vidas.

3.3. Un nuevo entretenimiento: los videojuegos

Durante la década de 1.970 comienza a aparecer un nuevo tipo de entretenimiento que hace furor entre los más jóvenes: los videojuegos. La empresa Atari suele considerarse hoy la impulsora de aquella revolución, si bien las bases se habían establecido tiempo atrás.

Los verdaderos inicios se remontan a la década de los 50 e incluso antes. No está claro a quién atribuir el honor de haber inventado los videojuegos, debido principalmente a que no se llega a un acuerdo sobre la definición del propio término *videojuego* (cuyo uso, en cualquier caso, no se generalizó hasta la década de 1.970).

Si consideramos *videojuego* a un sistema que muestra juegos *manipulando la señal de video* de un equipo de rasterización (como una televisión), entonces el mérito de la invención debemos dársela a Ralph Baer, quien patentó dicha idea en 1.966, aunque bajo el nombre de *"Television Gaming and*

Training Apparatus" (aparato de juego y entrenamiento para la televisión). La idea terminó desembocando en *Odyssey*, que es considerada la primera *consola de videojuegos* para el hogar, surgida en 1.972 (figura 3.1d)[3]. El 13 de Febrero de 2.006, Baer recibió la Medalla Nacional de Tecnología de la mano de George W. Bush por su invención[4].

Por otro lado, si pensamos en los *juegos de ordenador*, es decir aquellos creados gracias a la "programación" de un ordenador (digital o analógico), entonces hay que remontarse a 1.947. Ese año se creó el primer *juego electrónico interactivo* que simulaba el lanzamiento de un misíl, inspirado en los radares de la aún humeante Segunda Guerra Mundial. También son conocidos OXO (un "tres en raya" para EDSAC de 1.952, figura 3.1a), "Tennis for two" ("Tenis para dos", de 1.958, figura 3.1b) o Spacewar (de 1.961, figura 3.1c)[5] muy parecido al posterior Asteroids.

El éxito de este nuevo modo de entretenimiento fue explosivo. La historia de Atari atestigua esto desde sus inicios. Cuando tuvieron un prototipo de su primer videojuego, pensado para competir con otras máquinas de ocio como billares, futbolines o *pinballs*, decidieron probarlo en un local de su ciudad. Al día siguiente el dueño les llamó diciéndoles que su máquina se había roto. Al llegar, desanimados, para ver qué había ocurrido vieron que la causa fue que el cajetín de las monedas se había llenado, lo que impedía que la gente pudiera seguir jugando.

En 1.977 se comercializó la Atari 2600, primera videoconsola de cartuchos, que fue un completo éxito, disparando las primeras alarmas sobre los posibles efectos en la conducta infantil de estos sistemas. Desde entonces, las características de estos artilugios se ha ido incrementando a la misma velocidad que el resto del *hardware* informático[6], llegando a dejar obsoleta la definición dada previamente sobre el término *videojuego*.

En realidad, dicha definición es demasiado restrictiva, y más bien parece que fue un poco "forzada" por Ralph Baer para poder ser reconocido como el inventor del género[7]. Hoy en día, el término *videojuego* resulta mucho más amplio. Generalmente, se utiliza para referirse a cualquier juego que requiere *interactividad* por parte del usuario (jugador) a través de algún tipo de interfaz, y que genera el resultado en una pantalla, y como tal lo utilizaremos a partir de ahora.

[3] `http://www.ralphbaer.com/how_video_games.htm`

[4] `http://www.whitehouse.gov/news/releases/2006/02/images/20060213_d-0217-515h.html`

[5] `http://www.pong-story.com/intro.htm`

[6] En concreto, se considera que las consolas actuales (XBox 360, Play Station 3 y Wii) pertenecen a la *séptima generación*.

[7] A pesar de esto, no puede negarse la importancia de su contribución. Aunque es posible que la idea de un juego electrónico interactivo hubiera surgido antes de que él patentara su idea, es claro que él sí se esforzó por *comercializarla*, cosa que los anteriores no habían hecho.

Como se ha esbozado, hemos "heredado" el prefijo *video-* de los orígenes del área, cuando esa pantalla era de "rasterización" (típicamente una televisión o un monitor[8]). Sin embargo, hoy el espectro se ha ampliado increíblemente, pues incluye muchas otras plataformas, como los juegos de ordenador, de consolas, para móviles o para PDA (*Personal Digital Assistant*, Ayudante Digital Personal)[9]. En general, integran audio y vídeo, y facilitan experiencias que resultarían muy difíciles de vivir en la realidad.

3.3.1. Géneros de videojuegos

La cantidad de posibilidades es tan grande que es habitual realizar clasificaciones tanto en la *plataforma* como en el *género*. Respecto al primer aspecto, ya se ha comentado ligeramente en el párrafo anterior. Por plataforma se entiende el "soporte" y "entorno" sobre el que se ejecuta el videojuego, encontrando así los ordenadores, consolas, móviles o PDA. A veces se incluye aquí una quinta "plataforma" que recibe el nombre genérico de *Internet*, para referirse a aquellos juegos que son accedidos y utilizados directamente a través de la Web. En principio, no encajan en ninguno de los otros cuatro tipos por su carácter *multiplataforma* (suelen estar hechos en Macromedia Flash o en Java) lo que les permite, de hecho, ser jugados en virtualmente cualquiera de ellas.

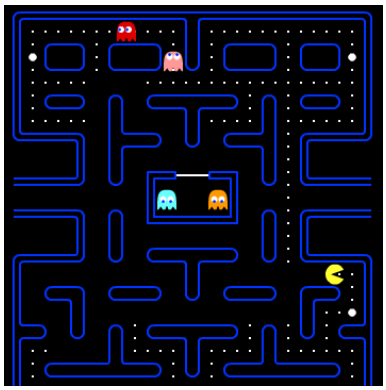
En lo que se refiere al *contenido* de los videojuegos, es decir a su "tipo", podemos encontrar varios grandes géneros. En realidad, intentar proporcionar una lista fiel sobre los géneros existentes es una labor sin final feliz: no hay un consenso en la industria sobre una definición formal universalmente aceptada sobre los géneros. Además, dado que el ámbito tecnológico se mueve muy deprisa, algunos géneros considerados hace 10 ó 20 años han dejado de tener sentido, o han sido absorbidos por otros. Por ejemplo, cualquier clasificación realizada durante la década de 1.980 incluiría guerras espaciales o "matamarcianos" (Asteroids, Space Invaders o R-Type), juegos de laberintos (Pac-Man, Guzzler, Mad Mix Game), de puzzles (Tetris, Columns, Lemmings) o "beat'em up" (Double Dragon, Golden Axe, Renegade, Ikari Warriors). Sin embargo hoy todos entran dentro de la categoría de *arcades*[10].

Por si esto fuera poco, en realidad la mayor parte de los juegos de hoy en día no encajan únicamente bajo un género, sino que éstos aparecen mezclados en un sólo producto. En cualquier caso, y a riesgo de levantar discrepancias, se enumera a continuación una posible lista de géneros de videojuegos:

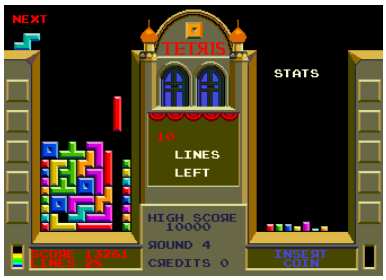
[8] Inicialmente el término monitor se utilizaba para referirse a "una televisión sin sintonizador"

[9] Hoy en día la penetración de los videojuegos es tal que incluso las aerolíneas los han integrado en la "oferta de ocio" proporcionado a los viajeros en los vuelos de larga distancia.

[10] En este sentido resulta curioso el capítulo 3 del libro de Crawford (1982) titulado "A Taxonomy of Computer Games" que, visto con la perspectiva que da el tiempo, demuestra que, efectivamente, las clasificaciones de los géneros de juegos son volátiles.



(a) Pacman



(b) Tetris

(c) Doom

(d) Half-Life

Figura 3.2: Juegos de arcade y de acción

- *Arcade*: suelen ser juegos cuyos principales ingredientes son *la habilidad* y la *rapidez de reacción*, dejando de lado la estrategia y la necesidad de razonar. Como hemos comentado, la mayor parte de los primeros videojuegos pertenecen a esta categoría (Pacman figura 3.2a, Donkey Kong, Space Invaders, Arkanoid, Bubble Bobble, Tetris, figura 3.2b). También se pueden encuadrar aquí los conocidos como *videojuegos de plataformas*, en los que el personaje controlado por el jugador debe avanzar evitando obstáculos y enemigos, moviéndose entre diferentes plataformas (Sonic, Super Mario Bros., Mega Man, Commander Keen, Príncipe de Persia original). Hoy este subgénero ha conseguido sobrevivir al salto a 3D, aunque con cierta dificultad, seguramente por la complejidad de estimar las distancias que ocasiona la tercera dimensión (Rayman, nueva saga de Príncipe de Persia, Blynx).

- *Acción*: son en cierto modo la evolución de parte de los juegos de arcade. La jugabilidad se basa en acciones rápidas en tiempo real. Los FPS (*First Person Shooters*, Acción en primera persona) son quizá los más representativos (Wolfenstein 3D, Doom figura 3.2c, Quake, Half-Life figura 3.2d, Unreal Tournament, Halo, Call of Duty).

(a) Monkey Island

(b) SimCity

(c) Los Sims

(d) Wii Sports

Figura 3.3: Juegos de aventura, simulación y deportes

- *Aventura*: se centran en la *narración* más que en cualquier otra cosa. Esta fuerte dependencia de la historia hace que beban de fuentes como la literatura y el cine, por lo que heredan los géneros literarios (ciencia ficción, fantasía, misterio, horror, comedia...). De hecho, son considerados un modo de *narración interactiva* en la que el jugador interfiere con el discurrir de la historia. Originalmente se conocían como *juegos conversacionales* al basarse en descripciones textuales de la situación sobre las que el usuario interactuaba con verbos sencillos (Colossal Cave Adventure, Zork), aunque durante la década de 1.980 se comenzó a acompañar esas descripciones con imágenes (Don Quijote, Mystery House). Con el tiempo, la parte visual comenzó a desplazar al texto, primero tímidamente (King's Quest, Leisure Suit Larry), y con la llegada de LucasArts y su sistema SCUMM de una manera más notable al pasar a utilizar únicamente el ratón como dispositivo de entrada (Maniac Mansion, Monkey Island figura 3.3a, Indiana Jones). Esto supuso también un cambio de nombre, al empezar a ser conocidas como *aventuras gráficas*. El modo de interacción siguió evolucionando, aunque nunca olvidó el énfasis en la historia y su narración (Blade Runner, Grim Fandango). No obstante, hoy el género se ha diluido y mezclado con otros.

- *Simulación*: tratan de recrear una *experiencia* donde aplicar conocimientos específicos. Algunos requieren, de hecho, una buena fase ini-

cial de preparación y aprendizaje, aunque otros resultan mucho más sencillos o cuentan con tutoriales iniciales. El ejemplo más claro lo tenemos en los simuladores de vuelo (F-19 Retaliator, Microsoft Flight Simulator), aunque los hay de otros dispositivos (submarinos en Silent Hunter, o trenes en Microsoft Train Simulator). Dentro de este género se encuentran los juegos conocidos como *God Games* en los que el jugador "juega a ser Dios" controlando ciudades (SimCity, figura 3.3b) o a personas (Los Sims, figura 3.3c).

- *Deportivos*: es uno de los clásicos, especialmente si recordamos el ya mencionado *Tennis for two* o el Pong. A veces se les considera integrados en el género de simulación, dado que recrean el juego de deportes tradicionales como fútbol (FIFA, Kick Off, Emilio Butragueño, PC-Futbol), baloncesto (NBA, Fernando Martín), Golf (Lynx, Wii Sports) o tenis (Emilio Sánchez Vicario, Pro Tennis WTA, Wii Sports figura 3.3d). También aquí podríamos colocar juegos de lucha (muy escasos hoy en día, mezclados con los juegos de acción) o de carreras (Moto GP, V-Rally).

- *Estrategia*: promueven la planificación, la gestión de recursos, la creación de hipótesis, el razonamiento y la posterior toma de decisiones. Los más conocidos son los asociados a situaciones bélicas (Command and Conquer, StarCraft, Praetorians figura 3.4a) o al crecimiento de civilizaciones (Age of Empires, Civilization figura 3.4b). También hay algunos intentos de mezclar ambas alternativas (Imperial Glory). Dado que el usuario toma decisiones que afectan a un entorno recreado por el ordenador, a veces se les encasilla dentro del género de la simulación.

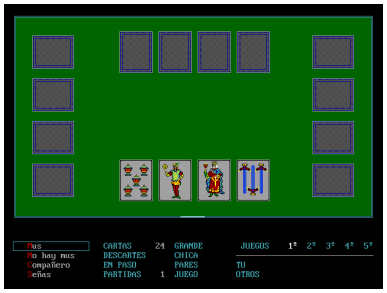
- *Rol*: al igual que los de aventura, suelen dar importancia a una historia, pero poniendo un énfasis especial en la evolución de los personajes. Se inspiran en los juegos de rol tradicionales jugados con lápiz, papel y dados, aunque llevados al mundo digital. Esto ahorra tiempo al ser el sistema el que se encarga de seguir la pista de las complejas reglas y abre otras nuevas posibilidades. Hoy resultan especialmente significativos los juegos de rol masivos on-line con mundos persistentes jugados por infinidad de personas simultáneamente a lo largo y ancho del mundo, con sus propias reglas socio-económicas. Ejemplos de este género son Final Fantasy, Ultima, Neverwinter Nights o World of Warcraft (figura 3.4c) por mencionar sólo algunos. Dada la importancia que se le da a la historia, el género suele mezclarse con los juegos de aventura. Además, como los jugadores normalmente persiguen objetivos ambiciosos para mejorar el "nivel" de sus personajes utilizando todo tipo de estratagemas, también pueden encajar dentro del género de estrategia.

- *Mesa*: son encarnaciones digitales de juegos clásicos de tablero, como

(a) Praetorians

(b) Civilization III

(c) Word of Warcraft



(d) Ordago

Figura 3.4: Juegos de estrategia, rol y de mesa

ajedrez, damas, tres en ralla, othello, dominó o go. También en esta categoría entrarían la multitud de juegos y solitarios de cartas (mus figura 3.4d, poker, carta blanca). Hoy muchos de ellos son jugados online.

La clasificación descrita aquí es una más entre la infinidad de opiniones existentes. Como ya se ha dicho, la mezcla de géneros y la aparición de otros nuevos hace imposible la realización de una taxonomía aceptada por todos. Como muestra de las discrepancias, es suficiente con comparar las categorías propuestas por Wolf (2002) en su capítulo 6 ("Genre and the Video Game") y la taxonomía de Apperley (2006). Este último considera únicamente cuatro grandes géneros (simulación, estrategia, acción y rol) mientras que el primero describe nada más y nada menos que cuarenta y uno (entre los que se encuentran aventura, vida artificial, tablero, combate, conducción, laberintos, pinball o rol por nombrar sólo unos pocos).

3.3.2. Los videojuegos en cifras

La penetración del mundo de los videojuegos en la sociedad actual es significativa. ADESE (Asociación Española de Distribuidores y Editores de Software de Entretenimiento) publica con regularidad estadísticas sobre los

hábitos de uso de este tipo de entretenimiento en España, y los resultados hablan por sí mismos. En Junio de 2.006 el número de jugadores de videojuegos (incluyendo PC, consolas y móviles[11]) alcanzó los 8'8 millones, lo que constituye el 20 % de la población total. Sólo la mitad juega sobre una única plataforma; el resto reconoce jugar en al menos dos de ellas, o incluso en las tres. De hecho, se estima que *uno de cada tres hogares* dispone de, al menos, una consola de videojuegos.

Resulta, además, curioso que estas consolas y los ordenadores estén cada vez más presentes en hogares *sin hijos*. Se está de hecho produciendo un desplazamiento en la edad media de los jugadores de videojuegos. Originalmente éstos eran catalogados como un entretenimiento dirigido al mundo infantil e informático; sin embargo aquellos niños que se sintieron "enganchados" por este modo de ocio han crecido y continúan con su afición. Si bien la presencia de jugadores es mucho más significativa entre los niños de 11 a 16 años (el 78 % de ellos son, en menor o mayor medida, jugadores de videojuegos), el perfil del jugador no es tan juvenil. Cuatro de cada diez jugadores tienen entre 20 y 34 años[12]. En Estados Unidos este incremento en la edad de los jugadores es aún más destacada. Según la ESA (*Entertainment Software Association*, Asociación de software de entretenimiento), la edad media de los jugadores es de 33 años y sólo el 31 % están por debajo de los 18[13]. Esto ha hecho evolucionar el propio mercado, que ha debido adaptarse a los gustos de los adultos aunque, naturalmente, sin olvidarse de los más jóvenes.

En lo que se refiere al género, según ADESE los videojuegos son una cuestión principalmente masculina. El 62'2 % *de los jugadores* son hombres[14], y tan sólo el 14 % de las mujeres españolas dicen utilizar videojuegos. Este resultado es muy diferente al obtenido por un estudio de la Universidad Europea de Madrid, publicado en Diciembre de 2.006 sobre la relación entre mujeres y videojuegos[15]. Aseguran que más de la mitad de las mujeres encuestadas emplean su ocio en los videojuegos de manera asidua. A pesar de estas discrepancias, lo que sí parece claro es que el porcentaje de jugadoras está experimentando una tendencia al alza.

Todo lo anterior muestra que la penetración de los videojuegos es, efectivamente, significativa. Esto tiene una repercusión obvia en la economía del ocio. Ellis (1983) indica que en Estados Unidos en el año 1.982 eran suficiente seis semanas para amortizar el precio de una máquina recreativa. También relacionado con la economía, las estimaciones del consumo de ocio

[11] La estadística sólo considera "jugador de móvil" a aquél que se ha descargado juegos adicionales a los predefinidos que se proporcionan con el propio teléfono.

[12] La media de edad es de 22 años.

[13] `http://www.theesa.com/facts/gamer_data.php`

[14] La ESA informa de un resultado muy similar; en Estados Unidos, el 62 % de los videojugadores son hombres.

[15] `http://www.uem.es/web/cin/cin2/observatorio/EstudioMujeresyvideojuegos.PDF`

audiovisual e interactivo en España en 2.005 fueron:

- Películas de vídeo: 470 millones de Euros (19 %)
- Música grabada: 480 millones (20 %)
- Taquilla cine: 627 millones (26 %)
- Videojuegos: 863 millones (35 %), de los cuales 326 millones se gastaron en hardware y 537 en software.

Esto muestra que el área de los videojuegos *supera* al dinero invertido por los españoles en el cine en salas. Durante *el primer día* que estuvo a la venta en Estados Unidos la consola Play Station 2 en 2.001, se alcanzaron los 150 millones de dólares en ventas. Esto es *seis veces más* que la recaudación del primer día de *La Guerra de las Galaxias: la amenza fantasma*, que llegó "tan sólo" a los 25 millones (Squire, 2003). Existe una muestra del *fenómeno social y cultural* en el que se han convertido los videojuegos en la transferencia de títulos. Tradicionalmente éstos han *licenciado* contenido de otros medios de ocio, como personajes de cine (Indiana Jones), películas (La Guerra de las Galaxias, Blade Runner a su vez basada en el libro "¿Sueñan los androides con ovejas eléctricas?" de Philip K. Dick), dibujos animados (los Simpsons), literatura (El Señor de los Anillos, Harry Potter) o cómic (Spiderman, e incluso Mortadelo y Filemón). Lo llamativo es que hoy en día, también ocurre en el sentido contrario. Así, se han realizado películas (por desgracia, de dudosa calidad) sobre los personajes Lara Croft y Pokemon, o sobre los juegos Alone in the Dark, Doom y Quake, estando en proyecto Hitman, Halo y Far Cry.

3.4. La era digital y el salto generacional

Entre padres e hijos siempre ha existido lo que normalmente se conoce como *salto generacional*, que hace referencia a las diferentes formas de ver o valorar las cosas. La causa principal es la diferencia de edad que provoca que padres e hijos estén en situaciones vitales completamente diferentes.

El paso del tiempo, sin embargo, termina poniendo a los hijos en el papel de padres, y las distancias desaparecen; suele decirse que uno se hace mayor cuando comienza a parecerse a su padre (o a su madre).

Prensky (2001) asegura, sin embargo, que actualmente el salto generacional es mucho más profundo de lo que solía ser *debido a las nuevas tecnologías*. Tradicionalmente, los hijos terminaban con opiniones parecidas a las de sus padres cuando el tiempo les llevaba hacia la edad adulta, dado que habían tenido una infancia y adolescencia *muy similar* a la de sus progenitores. Pero actualmente esto *no es así*. Las nuevas tecnologías *han moldeado* la forma

de pensar de niños y jóvenes, separándola de la de sus padres. El paso del tiempo situará a éstos en la etapa vital de sus padres, pero la forma de enfrentarse al mundo será (y de hecho ya es) diferente. Tal y como se ha mencionado previamente, el número de hogares *sin niños* con ordenadores y consolas se está incrementando año tras año: los padres de mañana quieren *para ellos mismos* un hogar diferente al que quieren sus padres.

Los niños de hoy, y los adultos de hasta 30 ó 35 años *han crecido* en un mundo diferente a aquél en el que lo hicieron sus padres. Han aprendido con Barrio Sésamo, que enseñaba a la vez que divertía, han sido asiduos consumidores de videojuegos, con los que han conducido, han sido alcaldes, resuelto misterios, luchado cuerpo a cuerpo, capitaneado tropas o pilotado aviones. Seguramente, por el camino, hayan perdido la oportunidad de *leer libros*, algo que sus padres quizá hicieran más; pero las vivencias que han tenido no por ello han sido más pobres. Según Carreño de la Cruz y Merino Malillos (2006), tradicionalmente los niños se socializaban en espacios presenciales, de ocio, formales y familiares. Los jóvenes tienen ahora un espacio de socialización adicional gracias a las TIC (Tecnologías de la Información y la Comunicación).

Así, la televisión, el mando a distancia, el teléfono, Internet, la mensajería instantánea, los SMS (*Short Message Service*, Servicio de Mensajes Cortos) y la Web les han hecho crecer en la *era digital*, con sus nuevas formas de comunicación y de búsqueda de información que rompe completamente las referencias espacio-temporales tradicionales. Está tan inserta en sus vidas que es considerada parte natural del entorno; es difícil considerar algo *tecnología* si fue inventado *antes* de nacer y no se ha conocido el mundo sin ella. Hoy resulta humorístico el suceso que se produjo el 28 de Diciembre de 1.895, en la presentación pública del cinematógrafo, cuando parte del público huyó despavorido al ver una locomotora acercándose a ellos. Actualmente resulta inconcebible aquella inocencia, dado que *todos* hemos conocido el cine como una forma más de ocio. Sin embargo, eso no ocurre con los videojuegos y el resto de "tecnología". Es bien conocido el hecho de que los niños a menudo son capaces de manejar un DVD mejor que sus padres. Los "inmigrantes digitales" se sienten perdidos ante este "nuevo mundo", tan diferente al del lápiz y el papel en el que crecieron. El "homo digitalis", sin embargo, se siente cómodo en él, ve sus posibilidades, sus modos de uso y las formas de aprovecharlo sin dificultad. Para Beck y Wade (2004), se ha producido una discontinuidad cultural entre los videojugadores y sus padres.

En este sentido, Gee (2004) cree que el significado de la palabra *analfabetismo* debería ser revisado. En primer lugar, se entiende por *analfabeto* alguien que no sabe leer ni escribir. Detrás de esta definición hay demasiadas cosas implícitas. Un niño de 6 años sabe "leer" el periódico, pero *no lo entiende*. ¿Podemos considerarlo alfabetizado? La capacidad de "codificar y descodificar símbolos que representan palabras" no debería ser suficiente, da-

do que no es lo mismo leer la letra de una canción, una novela, un periódico o la constitución: es necesario tener conocimiento de fondo que sirva como contexto para la comprensión.

Esta deficiencia es en realidad antigua. Hoy en día va un poco más allá, dado que *no todo está escrito*; vivimos rodeados de símbolos que hay que saber interpretar para desenvolvernos en la sociedad (anuncios, señales, humor) o para ser capaces de entender los *textos multimodales* donde las figuras son igual de importantes (si no más) que el texto que las acompaña.

Las nuevas tecnologías añaden otra posible fuente de ignorancia adicional que resulta cada vez más crítica en la llamada *Sociedad de la Información*. Según Díez Calurano (2006), a día de hoy, el *analfabetismo digital* en España es equivalente al analfabetismo *en lectoescritura* en los años 60. Además, considera que para solventarla no es suficiente con enseñar las habilidades mecánicas en el manejo de ordenadores y otras tecnologías, al igual que, según Gee, no es suficiente conocer las letras para "saber leer". Es necesario conseguir que las personas sean algo más que meros consumidores de tecnología, y se conviertan en sujetos activos y conscientes del nuevo entorno tecnológico.

Desafortunadamente, la "brecha digital" es en realidad también una *brecha social*, que impide a gran cantidad de personas optar a conocimiento y puestos de trabajo que requieren habilidades con los ordenadores. Esta separación no es únicamente interna a los países desarrollados, sino que también ahonda la separación entre los diferentes pueblos. Por desgracia, cuando el Tercer Mundo consiga aprender a leer, seguirá lejos de los países desarrollados porque todavía tendrá que aprender a utilizar ordenadores[16]. Solventar este problema es el objetivo del proyecto OLPC (*One Laptop per Child*, Un portátil para cada niño)[17], que intenta conseguir fabricar portátiles de bajo coste (100 dólares norteamericanos) de características "reducidas" pero muy resistentes, pensados para los niños del tercer mundo. La intención es venderlos también en los países desarrollados al doble de precio (que seguiría siendo barato) para que cada compra financie uno en el Tercer Mundo. El proyecto no está exento de críticas, dado que hay otras cosas mucho más prioritarias que ésta y el dinero podría aprovecharse de otras maneras.

[16] Una forma curiosa de comprobar la penetración de esa "Sociedad de la Información" en los países del mundo es analizar el número de (las escasas) direcciones IP poseídas por cada país en función de la población. Así los países con más "IPs per cápita" son El Vaticano, Estados Unidos y Canadá (con más de dos IPs por habitante). En la cola se sitúan Nigeria y la República Democrática de El Congo, con una IP para cada 58.000 habitantes (Fuente: `http://www.modernlifeisrubbish.co.uk/article/ips-assigned-per-capita`)

[17] `http://www.laptop.org`

3.5. El potencial educativo de los videojuegos

La facilidad con la que los niños entran en la *era digital* tiene mucho que ver con la maleabilidad de sus jóvenes mentes. Pero, naturalmente, necesitan tener acceso a los ordenadores para poder comprenderlos, y conseguir así los conocimientos básicos para desenvolverse con las TIC. Esos conocimientos son conseguidos a través de la *educación informal*[18] que proporciona el uso habitual de ordenadores. En este sentido, Subrahmanyam y Greenfield (1998) y Carreño de la Cruz y Merino Malillos (2006) creen que buena parte de esta *alfabetización digital* se produce a partir *del uso de los videojuegos*, lo que, además, despierta el interés por la tecnología a una temprana edad. Así, al igual que los juegos motores permitían a los bebés familiarizarse con su entorno, los videojuegos se convierten en una excusa perfecta para que pierdan el miedo a las TIC y se acostumbren a ellas.

De hecho, los juegos de los mamíferos proporcionan un entorno controlado donde poner en práctica de un modo seguro las habilidades que se necesitarán durante el resto de la vida. Del mismo modo, los videojuegos proporcionan también un acercamiento distendido a las nuevas tecnologías. Lo interesante de esto estriba en que no sólo es aplicable a los niños, sino también a los adultos. Según Alonso y Gallego (2000), para muchos expertos el sencillo juego "Buscaminas", tradicional en los sistemas Windows, no persigue más que la familiaridad con el uso del ratón en los primeros momentos. Gracias a la práctica y la repetición constante, el cambio desde los interfaces basados en consola hacia los interfaces gráficos pudo realizarse de una manera más distendida[19].

Esto da un cierto carácter *educativo* a los juegos de ordenador, puesto que, aunque pueda parecer un tanto pobre, no dejan de ser una forma de enseñar conocimiento informal. Resulta tentador intentar encasillar el modo en el que "enseñan" en función de las teorías del aprendizaje descritas en la sección 2.4. La mayor parte de los juegos más sencillos (como el buscaminas mencionado antes, o muchos de los tradicionales *arcades*) se basan en ensayo y error de modo que los usuarios aprenden lo que deben o no deben hacer. Es por lo tanto, tal y como nos dicen Carreño de la Cruz y Merino Malillos (2006), una aproximación conductista de corte skinneriano. Lo interesante de esta mezcla consiste en que la fórmula de refuerzos a través de premios o castigos (esto es, del condicionamiento operante) se basa en sumar o restar puntos, en la obligación de comenzar de nuevo, en pasar a nuevos (¡y más difíciles!) niveles,

[18] En la sección 2.4 la definíamos como cualquier actividad que genera efectos educativos pero carece de una estructura y organización que determine exactamente los objetivos que persigue.

[19] En realidad Díez Calurano (2006) comenta que alfabetizar tecnológicamente es mucho más que difundir el uso de las tecnologías. Hace falta superar la falta de interés, el desconocimiento y la ausencia de formación. En este sentido el buscaminas pretendía facilitar el uso del ratón, pero no está claro de que consiguiera generar interés por los ordenadores.

o en el mero hecho de competir, ya sea contra otros jugadores (humanos) o contra la propia máquina.

En realidad esta necesidad de conseguir ser capaz de hacer algo "complejo" a fuerza de repetir no es una cuestión única de los videojuegos. Hay muchos otros juegos tradicionales que son igual de repetitivos y que aun así se consideran (o se han considerado) divertidos, como el juego de la rana, la petanca, los dardos, los bolos o, de hecho, un gran número de deportes.

En *videojuegos* más complejos, las capacidades intelectuales necesarias son más exigentes, y el ensayo y error no es suficiente. Siang y Rao (2003) consideran que el aprendizaje mediante el condicionamiento operante de Skinner sigue existiendo aún en estos juegos, pero sólo durante las primeras interacciones. En ellas, el usuario aprende las reglas básicas del juego y se acostumbra al tipo de estímulos que provoca. A partir de los resultados inmediatos obtenidos (positivos o negativos), el usuario es capaz de reaccionar rápidamente al tipo de interacción y las reglas que dominan el juego. Sobre ellas se empezará a razonar de manera consciente para encontrar la mejor forma de responder a una situación nueva, retocar lo previamente aprendido en función de los nuevos hallazgos, etcétera. Este nuevo tipo de planteamiento se aproxima por lo tanto a las teorías cognitivistas del aprendizaje. De esta forma, los videojuegos potencian el aprendizaje por descubrimiento una vez que, quizá a nivel subconsciente, se han aprendido las reglas básicas y las interacciones mediante prueba y error.

Por otro lado, resulta cuanto menos sorprendente la *motivación interna* que la competición y la puntuación ocasiona a los jugadores, que se identifican con los protagonistas del juego e intentan superarse a sí mismos. Conseguir este interés por la mejora resulta mucho más difícil en la enseñanza tradicional. Naturalmente, las razones que hacen los videojuegos *divertidos* no son claras, de ahí que se sigan realizando juegos que no tienen éxito. No obstante, ha sido motivo de investigación desde hace mucho tiempo. Quizá fuera Malone el primero que se lo planteó, concluyendo, tras una serie de encuestas, entrevistas y observaciones propias, que para hacer un juego divertido se necesitaban tres ingredientes: reto, fantasía y curiosidad (Malone, 1981a,b).

También Gee (2004) muestra su sorpresa respecto a la gran capacidad de motivación de los videojuegos a pesar de su increíble dificultad que, además, sigue una tendencia creciente. Para los no jugadores o los jugadores esporádicos (*casual gamers* en el término anglosajón) esto resulta completamente inverosímil. Sin embargo, la cada vez mayor dificultad de los juegos indica que los consumidores así lo reclaman, pues de otra manera los juegos difíciles no se venderían y los desarrolladores no los harían (Gee hace un símil aquí con la teoría de la evolución de Darwin). Lo verdaderamente llamativo de este hecho es que buena parte de los usuarios de estos juegos son a la vez estudiantes de nuestras escuelas e institutos, que tienen una tendencia

completamente opuesta. Los alumnos *no aprueban*, y como resultado las instituciones bajan el nivel y hacen que los planes de estudio sean cada vez más y más fáciles. En la sección 3.7 volveremos sobre este tema.

En cualquier caso, anteriormente se habló de la capacidad que tienen los videojuegos en la *alfabetización digital*, la mayor parte gracias a la mejora de lo que a menudo se conoce como *coordinación ojo-mano*. Esto ha sido analizado de manera reiterada. Por ejemplo, Perzov y Kozminsky hicieron un tempranero estudio con niños israelíes en 1989. Por su parte, en el estudio realizado por Pérez Martín y Ruíz (2006) se concluyó que el 67'8 % de los jugadores afirma que los videojuegos han influido en la mejora de su destreza visual. Debido a que esto es relativamente fácil de medir de manera objetiva, realizaron un experimento de tiempo de respuesta ante un estímulo visual y comprobaron que, efectivamente, en general los usuarios de videojuegos respondían de manera más certeza que los no jugadores. Green y Bavelier (2007) han hecho un estudio similar algo más cuidadoso. Durante un mes tuvieron a un grupo de personas jugando una hora diaria a un videojuego de acción (Unreal Tournament), y mostraron una mejora del 20 % respecto al grupo de control en su habilidad para identificar letras colocadas de manera arbitraria[20], una prueba de agudeza visual habitual realizada por oftalmólogos. Lo curioso del experimento es que dicho grupo de control también jugó una hora diaria, pero no a un FPS, sino al famoso Tetris, juego que, en principio, parecería que permite practicar la agudeza visual de un modo más intenso que los juegos de acción.

De todas formas, es ampliamente reconocido que incluso los juegos más sencillos son capaces de enseñar algunas otras cosas interesantes. Por ejemplo, Gee (2004) cree que enseñan a experimentar el mundo de un modo nuevo y diferente. Por su parte, ADESE asegura que el 77 % de los usuarios consideran que los videojuegos potencian la competitividad[21]. Relacionado con esto, Pérez Martín y Ruíz encontraron que el 53'62 % de los jugadores afirmaron que los videojuegos habían influido *mucho* en la mejora de su capacidad de superación, aunque este porcentaje era bastante irregular según la edad, siendo mucho mayor en los jóvenes y quedándose tan sólo en un 30'15 % entre los mayores de 35 años. Por el contrario, tan sólo el 10'84 % creía que los videojuegos no les habían influido en absoluto en lo que se refiere a esta capacidad de superación.

ADESE continúa diciendo que el 76 % de los jugadores cree que mejora la agilidad mental y el 49 % la creatividad. En este sentido, Gee (2004) opina que los videojuegos fuerzan a desarrollar e interiorizar recursos para aprendizajes futuros y para resolución de problemas variados.

[20] En concreto, se les pedía que identificaran con rapidez la orientación de una letra T situada entre otros símbolos cuya única finalidad era despistar.

[21] Éste y el resto de estimaciones proporcionadas por ADESE están publicadas en la sección "Realidades" de `http://www.adese.es/web/informes.asp`.

Curiosamente, según ADESE sólo el 45 % de los jugadores considera que ayudan a mejorar los conocimientos sobre informática. Resulta también llamativo que el 53 % opine que apoyan el aprendizaje de idiomas. Esto es en realidad más una demostración del quizá bajo número de traducciones al español de los juegos internacionales que de una ventaja directa e intencionada de los propios juegos.

El 71 % de los usuarios prefieren jugar en grupo. Esto entra en conflicto con el dato de que tan sólo el 35 % considera a los videojuegos como una actividad bastante social. También Pérez Martín y Ruíz (2006) se preocuparon por esta parte social. El 41 % de los jugadores encuestados afirmaron haber hecho amigos a través de los videojuegos. Además, los videojuegos, aseguraron, habían influido mucho en su capacidad de trabajar en equipo. Este resultado corre paralelo con los hábitos en el modo de jugar en grupo. El 65 % dice distribuir las tareas a la hora de conseguir objetivos en el juego, y sólo el 16 % afronta el juego de manera individual. En principio podría pensarse que esto último sólo tiene sentido en el caso de los juegos multijugador. Sin embargo, en general los videojuegos promueven el aprendizaje colaborativo[22] incluso aunque sea monousuario. Esto se debe a que aunque se juegue solo, normalmente la propia actividad generará conversaciones a su alrededor entre hermanos o amigos para buscar ayuda. Desde la llegada de Internet este carácter social es todavía más destacado, ya que a menudo se crean grandes comunidades alrededor de los videojuegos de mayor éxito que proporcionan ayuda y pistas. Los jugadores *aprenden* a superar los juegos de maneras nuevas e inverosímiles sin necesidad de una *enseñanza reglada* sobre él (ni siquiera se suele leer el manual, después de todo).

La lectura que Gee (2004, página 45) hace de este nuevo modo de "socializarse" consiste en afirmar que los videojuegos proporcionan el potencial para unirse y colaborar en un nuevo grupo de afinidad. Esto tiene una importancia mayor de la que en principio podría pensarse. Formar parte de uno de estos grupos significa participar en su lenguaje, forma de pensar y opiniones. Es, de hecho, el aspecto que faltaba en la definición tradicional de *alfabetismo*: para poder leer (y entender) escritos sobre baloncesto hay que pertenecer al grupo de aficionados a este deporte. Permitir a los niños y jóvenes pertenecer a un grupo de afinidad les ayuda a absorber la mentalidad del grupo, a disfrutar de él al mismo tiempo que son conscientes de la posibilidad de ser manipulados y de manipular a otros gracias a las relaciones en esa "sociedad". Algunas de las conclusiones que de manera lateral podrían aprenderse tienen relación con la justicia, la caridad, la habilidad para saber su posición dentro del grupo, etcétera.

Quizá podría pensarse que fomentar la capacidad de superación, mejorar la agilidad mental o potenciar el trabajo en grupo no es, después de todo,

[22] En este caso, nos estamos refiriendo al aprendizaje sobre el propio juego, sus reglas y modo de superarlo, no al posible aprendizaje de otros dominios.

enseñar. No obstante, en la sección 2.4 decíamos que la educación pretendía el perfeccionamiento de la persona en cualquiera de sus dimensiones (intelectual, física, estética, social, profesional, ética y moral, ...). Por tanto, aunque las capacidades descritas aquí no tengan una relación directa con las materias tradicionalmente enseñadas en la educación reglada[23], no significa que no podamos considerar a los juegos como educativos, al menos en ciertas dimensiones.

De hecho, se han levantado varias voces avisando de la importancia de todas esas habilidades. John Seely Brown (exdirector de Xerox PARC) opina que algunas de las habilidades que serán valoradas por las empresas del futuro se estimulan y premian más en los videojuegos que en las clases tradicionales. En su controvertido libro, Beck y Wade (2004) creen que de hecho ese desplazamiento ya se ha producido. A raíz de una investigación en la que involucraron a más de 2.000 personas del mundo de los negocios, descubrieron aptitudes positivas en el trabajo (toma de riesgos o resolución de problemas) entre aquellos que crecieron entre videojuegos que no mostraron los demás.

Por otro lado, en su libro, Gunderson et al. (2004) plantean una serie de previsiones que vaticinan grandes cambios en el mercado laboral norteamericano. En concreto, el exsecretario de educación estadounidense, Richard Riley cree que los que serán los 10 trabajos más demandados en 2.010 *no existían* en 2.004. Además, se estima que los adultos de 38 años habrán tenido entre 10 y 14 puestos de trabajo diferentes, por lo que es necesario conseguir que los niños de hoy *aprendan a aprender*. La capacidad de superación y la búsqueda no guiada de información impulsada por los videojuegos podría ser la mejor manera de prepararles para enfrentarse al futuro que les espera.

Karl Fisch, un profesor de Colorado, deduce, a partir de todo lo anterior, que actualmente estamos formando estudiantes para trabajos que aún no existen, que tendrán que hacer uso de tecnologías que todavía no han sido inventadas, para resolver problemas que hoy no hemos reconocido como tales[24].

3.6. Críticas comunes a los videojuegos

Por desgracia, los videojuegos han sido ignorados de manera reiterada por los docentes (Squire, 2003). Y, las pocas veces que se les ha prestado atención, ha sido para centrarse en sus consecuencias *sociales*, ignorando el inmenso potencial educativo.

De hecho, a pesar de la visión optimista que hemos intentado proporcionar de los videojuegos, hoy en día es aún difícil conseguir que la sociedad se

[23] Más adelante repasaremos, no obstante, las posibilidades de los videojuegos para enseñar cuestiones del currículo.

[24] `http://thefischbowl.blogspot.com/2006/08/did-you-know.html`

conciencie de todos estos aspectos positivos. La causa principal es la mala imagen que a menudo los diferentes medios de comunicación crean sobre esta forma de ocio, creando así una inercia negativa contra la que sólo el tiempo es capaz de luchar. Cuando en los medios generalistas se habla de videojuegos, la mayor parte de las veces es para destacar su contenido violento o los trastornos de personalidad que, dicen, pueden generar.

En realidad, según Gee (2004) incluso los videojuegos violentos tienen cosas positivas que enseñar. En cualquier caso, según ADESE la mayoría de los usuarios considera que los contenidos de los videojuegos tienen un nivel de violencia similar al de otras formas de entretenimiento y no debe ser especial causa de alarmismo social. Del mismo modo, Pérez Martín y Ruíz (2006) nos dicen que los datos reflejan que, en todos los rangos de edad, resulta más fácil ver escenas de lucha, sangre o de muerte de un ser humano en televisión y el cine que en los videojuegos. De hecho, resulta curioso el estudio de Silvern y Williamson (1987) en el que encontraron que, efectivamente, los niños que habían jugado a *Space Invaders* eran algo más agresivos que los niños que no habían jugado, pero *no* había ninguna diferencia cuando se les comparaba con niños que habían visto dibujos animados en la televisión.

Por último, en relación a esto Gee (2004) opina que es el escándalo social causado por la violencia en los videojuegos es infundado, dado que esta agresividad no es otra cosa que un triste reflejo de la realidad. No obstante, excusa su reiterada aparición debido a que resulta muy fácil de programar. Pronostica que con el tiempo la violencia irá quedando al margen, según aparezcan nuevas formas de interacción. En realidad este desplazamiento es visible desde hace tiempo; tal vez hace una o dos décadas la mayor parte de los juegos de éxito fueran claramente agresivos, pero hoy se da mucha importancia a otros aspectos, principalmente a la historia.

En cualquier caso, es necesario procurar que haya un mayor control del acceso de los menores a este tipo de videojuegos. El código PEGI (*Pan European Game Information*, Información paneuropea sobre juegos) ha hecho mucho en este sentido (es, de hecho, más detallado que la clasificación por edades tradicional de las películas cinematográficas), pero aun es necesario concienciar a los padres *para que lo miren.*

Aunque el nivel de violencia suele ser el motivo principal de las poco informadas críticas, también hay otros aspectos que se miran con cautela. Seguramente las acusaciones más claras vinieran por parte de Provenzo (1991), quien opina que los videojuegos ocasionan violencia y agresividad, crean una imagen falseada de la mujer, e incluso aislamiento social.

Respecto a la parte de la violencia ya hemos hablado anteriormente. Por otro lado, el principal argumento esgrimido por Provenzo para demostrar la posibilidad de distorsionar la imagen de la mujer consistía en hacer notar que en muchos juegos el objetivo final era rescatar a una princesa, pareja o amiga del protagonista de las garras del enemigo. En realidad esta crítica

también resulta llamativa al comprobar que este hecho se produce también en otros lugares, como cuentos infantiles, dibujos animados o películas. Pero en cualquier caso, el *recurso* del "rescate de la chica" que era relativamente habitual en 1991, ha dejado de ser tan común hoy en día. Para una exploración en profundidad sobre la relación entre el género y los videojuegos puede consultarse el libro editado por Cassell y Jenkins (1998).

La crítica de Provenzo sobre el supuesto aislamiento social que sufren los videojugadores también ha perdido bastante peso. Ya hemos mencionado que desde el principio los videojuegos ocasionaban una razón adicional para el acercamiento entre individuos de gustos similares. Hoy con los juegos multijugador masivos esta realidad es todavía más patente. Por último, la gran mayoría (un 83 %) de los encuestados por Pérez Martín y Ruíz (2006) opinaron que su relación familiar *no* se ha visto alterada por el uso de este modo de entretenimiento. Por su parte, un 13 % consideró que, en realidad, había mejorado, y sólo un 3 % sintió que había empeorado.

Para una descripción mucho más exhaustiva de las razones que hoy invalidan las ideas de Provenzo puede consultarse Squire (2003). En cualquier caso, según The Economist (2005) las críticas a los videojuegos son en realidad una cuestión más generacional que verificada. Este problema no es nuevo. Ya se comentó en la sección 2.7 las pegas de Sócrates a la palabra escrita para la educación. También las novelas se consideraron literatura de insuficiente calidad para ser tenida en cuenta en las Universidades, o, más recientemente, el *Rock'n'Roll* fue catalogado como música diabólica que animaba a la violencia, la promiscuidad y el satanismo. Hoy en día la mayor parte de los videojugadores están por debajo de los 40 años, y la mayor parte de las críticas vienen de no jugadores que están por encima de esa edad.

3.7. Enseñanza intencionada a través de videojuegos

Aparte de las reacciones emocionales de disfrute, juego o poder que generan los videojuegos (Squire, 2003), la sección 3.5 ha demostrado que también ocasiona *aprendizaje*, aunque sea de manera no prevista o planeada de antemano.

Algo parecido ocurre en otros modos de ocio. El cine pronto se convirtió en una ventana a otros mundos, en una forma de condicionar a la ciudadanía sobre determinados temas, o de mostrarle cosas que no podría tener al alcance de otro modo. La televisión siguió en la misma linea, llevando a los hogares imágenes de países remotos, noticias sobre sucesos lejanos y, en definitiva, enseñando otras culturas.

Esta enseñanza como efecto lateral al ocio pronto se demostró poderosa y, como nos cuenta Rice (2005b), se le comenzó a dar peso con la aparición

de películas históricas, documentales o programas divulgativos y educativos. Algo parecido ocurre en el mundo de los videojuegos. Los profesores son conscientes del increíble interés que éstos suscitan en sus alumnos, por lo que resulta muy interesante tratar de canalizar dicho interés hacia el aprendizaje (Rice, 2005a). La intención por lo tanto es añadir contenido educativo en un videojuego para conseguir el tipo de enseñanza proporcionada por un sistema basado en marcos o, incluso mejor, un ITS, pero dentro de un videojuego y aprovecharnos de su motivación. Así, conseguiríamos las ventajas de la enseñanza individualizada planteadas por Bloom (1984), y ya argumentadas en la sección 2.5. Pero, además, dado que los juegos impulsan la competición y (a la vez) la cooperación y el trabajo en grupo también potencian otras dimensiones, especialmente la social, cosa que en general no ocurre con otros modos de enseñanza digital.

Curiosamente, esta idea de la *enseñanza intencionada* a través de los videojuegos surgió de manera bastante temprana. La patente que allá por 1.966 hizo Ralph Baer de su *"Television Gaming and Training Apparatus"* incluía, de hecho, la palabra *training* (entrenamiento o práctica) en el título. También proponía algunas clases de juegos (hoy llamados *géneros*, ya comentados en la sección 3.3.1) como juegos de acción, de mesa o deportivos. Entre su lista, aparecían ya los juegos educativos, ejemplificados por un posible juego para enseñar geometría o aritmética básica (por ejemplo, sumando bloques)[25].

Estas primeras y tímidas ideas cogieron fuerza con la gran popularidad de Pac-Man, a partir de la cual se planteó si no podría *embotellarse* su "magia" y llevarla a las clases para mejorar el disfrute y participación de los alumnos (Bowman, 1982). Ese mismo año, en su prematuro libro sobre diseño de videojuegos, Crawford (1982) hablaba tímidamente sobre los videojuegos educativos. En su capítulo 3 proponía su ya mencionada categorización de los géneros de los videojuegos, en el que hablaba de dos grandes bloques: juegos de acción o habilidad (de combate, deportivos, laberintos, etcétera), y los juegos de estrategia (aventura, rol, guerra, azar...). Entre estos últimos añade los videojuegos educativos, aunque, curiosamente, reconoce que, en cierto modo, todos lo son. Su característica principal es que están diseñados explícitamente con el objetivo de enseñar. Para desarrollarlos, los grupos dedicados al uso de ordenadores para enseñar necesitan disponer de un cierto interés por el diseño de videojuegos, algo que no siempre ocurre. El propio Crawford exploró este ámbito creando tres juegos educativos: *Energy Czar*, *Eastern Front (1941)* y *Scram*. El primero es un simulador para mostrar la relación entre energía, medio ambiente y economía. Por su parte, *Eastern Front (1941)* recrea la invasión de Rusia por parte de las fuerzas germanas del Este durante la Segunda Guerra Mundial. Y *Scram*[26], simula un reac-

[25] `http://www.ralphbaer.com/TypedNotes_page.htm`

[26] Textualmente "largarse" o "pirarse", es el nombre que recibe la parada de emergencia

tor nuclear que el usuario debe mantener en funcionamiento manipulando las válvulas e interruptores incluso en situaciones críticas como después de terremotos. A pesar de estar escrito en Atari BASIC, simulaba el reactor nuclear a través de ecuaciones diferenciales (Crawford, 2003, capítulos 16, 17 y 18). Los dos últimos fueron publicados a través de APX (*Atari Program Exchange*, Intercambio de Programas de Atari), un modelo de distribución poco común, en el que Atari animaba a los usuarios a crear software que luego era vendido a través de un catálogo por el que los creadores recibían parte de los beneficios a modo de derechos de autor.

Los ejemplos anteriores son en realidad antiguos. Si hacemos un análisis sobre los videojuegos actuales es fácil sorprenderse de su increíble dificultad, tal y como ya hemos mencionado. A modo de ejemplo, Bopp (2006) nos recuerda las instrucciones de Pong: "Inserte una moneda; pulse «Start»; evite no dar a la bola para conseguir mayor puntuación"[27], y las compara con el manual del juego *Falcon 4.0*, un simulador de vuelo de 1998 con sus casi 600 páginas.

Gee (2004) muestra su sorpresa al descubrir que los videojuegos actuales tienen una duración de entre 50 y 100 horas, y para conseguir terminarlo es necesario *aprender* cómo hacerlo: es necesario entender cómo usarlo, sus reglas explícitas e implícitas y las mejores estrategias. Ese aprendizaje no es en absoluto trivial, y a menudo resulta frustrante. Y sin embargo, muchos usuarios insisten una y otra vez, dedicándole horas y horas para conseguir exprimir el juego hasta las últimas consecuencias. A pesar de las dificultades, del esfuerzo y del tiempo, los jugadores disponen de suficiente motivación como para no cejar en el empeño. Lo increíble es que, a pesar de todo, el mercado parece pedir que los juegos sean difíciles, porque no dejan de venderse.

Esta observación resulta especialmente inquietante al compararse con la situación de los planes de estudio. Como ya esbozamos en la sección 3.5, los videojuegos son cada vez más difíciles y los adolescentes los devoran con avidez, mientras en clase se les da cada vez menos materia, se les exigen menos conocimientos mínimos porque, parece, cada vez se esfuerzan menos y consiguen peores resultados. Se baja el nivel para evitar frustración o inundación de suspensos mientras gran parte de su capacidad de aprendizaje se "derrocha" en los videojuegos. Henry Jenkins, profesor del MIT, lo resume diciendo que el peor comentario que se le puede hacer a una actividad extraescolar es que es demasiado difícil, mientras que en el caso de los videojuegos, lo peor es decir que es demasiado fácil[28].

de un reactor nuclear.

[27]"Insert coin", "Press start" y "Avoid missing ball for high score"

[28]FUENTE: Entrevista en "Games in Education", documental creado por Mark Wagner y Michael Guerena del *Orange County (CA) Department of Education's Educational Technology group*. Disponible en `http://video.google.com/videoplay?docid=6117726917684965691`

Esto dispara algunas preguntas sobre el dudoso éxito de varios siglos de pedagogía, en el que el hombre ha buscado la mejor forma de enseñar. Al reconocer que es necesario un aprendizaje para superar un determinado videojuego, es fácil darse cuenta de que las "teorías de aprendizaje" informales que aparecen integradas de manera natural en dichos videojuegos son muchísimo más efectivas que las propuestas por los pedagogos.

En realidad, no deberíamos ser tan injustos como para limitarnos a culpar a la pedagogía de la situación y criticarla por su falta de resultados. Al fin y al cabo, el modo de enseñanza tradicional ha funcionado durante mucho tiempo con relativo éxito. Quizá la verdadera causa esté en la *brecha generacional* causada por las nuevas tecnologías que se describió en la sección 3.4. Prensky (2001) asegura que la forma de pensar y enfrentarse al mundo de las nuevas generaciones es diferente. Los jóvenes de hoy no tienen ni peores ni mejores capacidades intelectuales que los de hace décadas: sencillamente están focalizadas en cuestiones diferentes (Howes, 2001). Hoy estamos acostumbrados a escuchar las noticias televisivas y *a la vez leer otras* que aparecen sobreimpresionadas en un texto que recorre la parte inferior de la pantalla con información sobre el tiempo, la cotización o cualquier otra cosa. Los jóvenes de hoy son capaces de mantener la atención *simultáneamente* en ambas cosas, aunque después no consigan centrarse y escuchar a su profesor hablando sobre geografía durante una hora. Están acostumbrados a aprender explorando mientras se divierten, pueden prestar atención a diferentes partes de la pantalla simultáneamente para evitar perder, o a encontrar el sentido a un vídeo musical que cambia de imagen cada segundo. El colegio no es difícil, *es aburrido*.

En este sentido, Bruno (1995) dice que los profesores consideran alumnos "de riesgo" a aquellos que muestran preferencias por actividades no dirigidas que consumen mucho tiempo, como ver la televisión y jugar a videojuegos. Mitchell y Savill-Smith (2004, página 13) hacen un recorrido más profundo por la bibliografía que relaciona resultados académicos y videojuegos. Relacionado con este tema, en Abril de 2.007 el gobierno chino ordenó a los operadores de juegos on-line que instalaran software *anti-adicción* en sus títulos, para impedir que los menores de 18 años pudieran jugar más de 3 horas diarias (Xing, 2007). La medida, conocida como *sistema anti-adicción de juegos on-line* supondrá que durante las tres primeras horas de juego éste funcionará de la manera habitual, durante las dos siguientes la puntuación conseguida será la mitad de la normal, y a partir de la quinta hora no se conseguirán puntos de ninguna forma y se mostrarán amenazas de pérdida total de los puntos acumulados.

Aunque, obviamente, ninguna adicción es buena, la posible relación entre uso de videojuegos y el empeoramiento en los resultados escolares puede ser leído de manera inversa. Si realmente son los videojuegos y la televisión los que provocan el cambio en la forma de pensar de los adolescentes, quizá su

uso ponga en evidencia las deficiencias de los sistemas educativos actuales. En ese caso, el problema no estribaría en que los videojuegos sean malos *per se*, sino más bien que despiertan en los usuarios nuevas inquietudes que convierten en *aburrida* a la escuela tradicional, lo que origina la pérdida de interés y el empeoramiento de los resultados.

Encontrar las razones de las diferencias resulta importante para intentar extraer lo mejor del mundo de los videojuegos e integrarlo en las clases regladas. Malone (1981b) hizo notar que en las clases tradicionales los alumnos tienen poco control sobre lo que aprenden: son receptores pasivos del material elegido por los profesores, tienen que adaptarse al nivel y velocidad del grupo, y reciben respuestas (*feedback*) de manera irregular e imprecisa, justo lo opuesto a lo que ocurre en muchos videojuegos. A partir de estas ideas, perfila varios elementos que los videojuegos educativos deberían tener:

- Objetivos que resulten claros y significativos al estudiante.
- Varios modos de conseguir el mismo objetivo, y una puntuación que proporcione información continua al alumno sobre su progreso.
- Diferentes niveles de dificultad que se adapten a la habilidad del usuario en cada momento.
- Elementos aleatorios de sorpresa que rompan la monotonía.
- Una historia de fondo que resulte interesante emotivamente, y que esté relacionada de algún modo con las habilidades que hay que ir consiguiendo durante el avance del juego.

Poco después Bowman (1982) llegó a unas conclusiones parecidas analizando experimentalmente a los jugadores de Pac-Man. Las habilidades necesarias y los retos propuestos en este *arcade* están, asegura, perfectamente equilibradas, la información proporcionada del progreso es inmediata, clara y perfectamente distinguible de otros estímulos banales.

Squire (2003) recoge todas estas ideas y se pregunta si los puntos fuertes anteriores de los videojuegos están en algún sitio dentro de la educación tradicional. El resultado, mostrado en la tabla 3.1, es bastante desolador, pues muestra la distancia existente entre el modo en el que se produce el aprendizaje dentro de un videojuego y la forma en la que se realiza en una clase tradicional. Esto es, quizá, normal si pensamos que los planes de estudio son diseñados por gente que vivió una niñez y adolescencia *muy diferente* a la de los jóvenes de hoy. Hay mucha inercia en los profesores e instituciones educativas que opinan que el aprendizaje debe *costar esfuerzo*, por lo que la mera mención de incluir diversión y juegos en la escuela levanta voces en contra. Thomas et al. (2003) opinan que este sentimiento negativo se debe a que el aprendizaje es considerado algo serio, y como tal sólo puede conseguirse con

Pac-Man	**Clase tradicional**
El jugador controla cuanto y cuando juega.	Los estudiantes, en grupo, aprenden al mismo paso y tienen poca libertad para controlar el contenido y la velocidad de aprendizaje.
Los jugadores se integran de manera activa en actividades rápidas, variadas y entretenidas.	Los estudiantes absorben de manera pasiva información en actividades rutinarias.
Los jugadores juegan y practican hasta que son buenos. Pueden estar tanto como sea necesario para conseguirlo.	Los estudiantes deben ir todos a la misma velocidad. La escuela tradicional coge un grupo de alumnos y fija el tiempo para el material que cada individuo debe dedicar, en lugar de mantener fijo el aprendizaje que cada uno consigue (y variar el tiempo).
Los jugadores *sienten* que son cada vez más hábiles dentro del juego	Los estudiantes aprenden conocimiento abstracto impuesto por los profesores, y lo *regurgitan* en el papel de examen sin haberlo aplicado.
En los videojuegos los usuarios trabajan juntos, compartiendo consejos y secretos	Los estudiantes trabajan de manera aislada, y no está permitido el intercambio de información, que se considera copia.
El rendimiento depende de cada uno; cada jugador compite contra su propia habilidad para mejorarla y alcanzar nuevos objetivos. Todos consiguen su propia visión de "nivel alto" en el juego, hasta que quedan satisfechos consigo mismos.	Los estudiantes son evaluados de manera normalizada, comparándose el rendimiento entre todos ellos y animándoles a competir entre sí.
Los juegos son usados por el premio intrínseco de jugarlos y el estado emocional que producen.	La escuela se organiza a través de premios extrínsecos cono buenas notas o miedo al fracaso.

Tabla 3.1: Comparación entre el Pac-man y una clase tradicional

herramientas igualmente serias[29]; parecería como si el aprendizaje entretenido fuera menos valioso que el conseguido con gran esfuerzo y dedicación.

En este sentido, Kafai (2001, 2006) se plantea una cuestión en cierto modo filosófica respecto a la idea de utilizar juegos para *suavizar* el aprendizaje. Absorber el conocimiento que la humanidad ha ido consiguiendo con esfuerzo es una labor ingente de una importancia capital. Kafai duda que los alumnos *valoren* los conocimientos si son aprendidos a través de juegos, convirtiendo algo tradicionalmente difícil (y valorado) por algo *divertido* y *fácil*.

Esta crítica tiene parte de razón. No obstante, conseguir que el aprendizaje sea *divertido* tiene un efecto positivo en la motivación intrínseca, es decir en el *deseo del alumno por ser enseñado*, lo que efectivamente parece mejorar el aprendizaje (Lepper y Cordova, 1992).

[29]Quizá sea para tratar de vencer estos prejuicios por lo que los videojuegos educativos empiezan a ser conocidos como *videojuegos serios*.

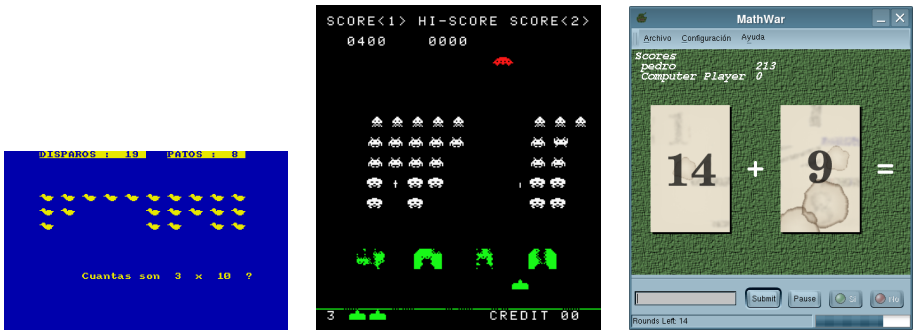
Respecto a la duda sobre hacer la educación más *fácil*, en realidad aquí hay un problema semántico de fondo. Rieber (1996) declara que hay una tendencia patente a definir *juego* como el antónimo de *trabajo*, pero es incorrecto. Cita a Blanchard y Cheska (1985) para hacer notar que, de hecho, no disponemos de un término exacto para definir lo contrario a *juego*. Además, la diversión no está reñida con la dificultad (¿no es complicado tirar 10 bolos con una sola bola, o encestar en una canasta situada a tres metros del suelo?).

Lo que queremos es conseguir que esa "difícil diversión" (*hard fun*) esté integrada en los sistemas educativos: no se busca necesariamente que el aprendizaje sea fácil, sino que sea *divertido*. Esto es, obviamente, lo que persiguen los *videojuegos educativos*, que buscan producir un aprendizaje *intencionado* más allá de las cuestiones como la capacidad de superación o la cooperación discutidas en la sección 3.5. La idea es crear *videojuegos educativos*, en los que la parte lúdica se mezcla con la pedagógica y el contenido que se desea que se practique o aprenda. Esto requiere la capacidad de *diseño de videojuegos* que ya anticipaba Crawford en 1982. El principal reto, tal y como nos cuentan Repenning y Lewis (2005), está en conseguir un correcto equilibrio entre ambos componentes (lúdico y educativo) para no terminar con un *software* consistente en un juego estropeado con preguntas, o en un programa educativo tristemente adornado con mecánicas de videojuego. Una unión inocente de entretenimiento y aprendizaje difícilmente conseguirá un resultado meritorio (Brown, 1995). Es necesario una unión *representativa* entre ambas partes, perfectamente integrada en el diseño del juego que dé un significado a las dos componentes de manera que una no tenga sentido sin la otra en el diseño del videojuego.

Como suele ocurrir, existe un *continuo* en el espacio que separa un videojuego de un programa educativo, y la virtud estará en el término medio. Para ejemplificarlo, se puede encontrar un paralelismo con la televisión y la publicidad. Las cadenas televisivas no ganan dinero poniendo películas y haciendo programas; el dinero lo ganan gracias a la publicidad. Los ciudadanos vemos la televisión porque nos entretienen sus programas o sus películas ($\simeq$videojuego) y, como efecto lateral, tenemos que ver publicidad ($\simeq$contenido educativo).

La peor unión posible es, de hecho, la más habitual: una película troceada con intermedios plagados de anuncios. El espectador pasa drásticamente de una parte entretenida a la obligación de esperar a que termine la publicidad. Es lo que Prensky (2001, página 25) denomina *"fuego y hielo"* debido a la contraposición de dos aspectos inversos de una manera tan radical.

La publicidad integrada directamente en los propios programas a modo de espacios o premios patrocinados es bastante parecida (y desagradable) al modelo anterior. La publicidad subliminal de productos que aparece en algunas series televisivas o programas de cocina son mucho más sutiles y,



(a) Tiro al Pato (b) Space Invaders (c) MathWar

Figura 3.5: Tiro al pato y MathWar

desde luego, preferibles. Pero, en cualquier caso, el grado de integración entre ambas partes que querríamos conseguir en nuestros videojuegos educativos es el conseguido en la película "Naufrago" (de Robert Zemeckis, estrenada en 2.001). El *guión* (≃diseño del videojuego) difícilmente tiene sentido sin la empresa de transportes (FedEx) y sin la marca deportiva fabricante del balón de voleibol (Wilson): la publicidad forma parte *inseparable* de la película. Lo mismo ocurre con la más antigua "Uno, dos, tres" de 1.961 a la que el propio Billy Wilder, su director, sarcásticamente catalogó como el anuncio más largo de Coca-Cola.

Sólo consiguiendo esta integración perfecta en el diseño de un videojuego educativo conseguiremos la *motivación intrínseca* (Malone, 1987). Como demostración del continuo entre un programa educativo y un juego, vamos a analizar diferentes videojuegos educativos reales dedicados al mismo dominio (*practicar* operaciones aritméticas) para ver las diferentes decisiones de diseño que colocarán a cada uno en un punto, más o menos cerca de cada extremo[30].

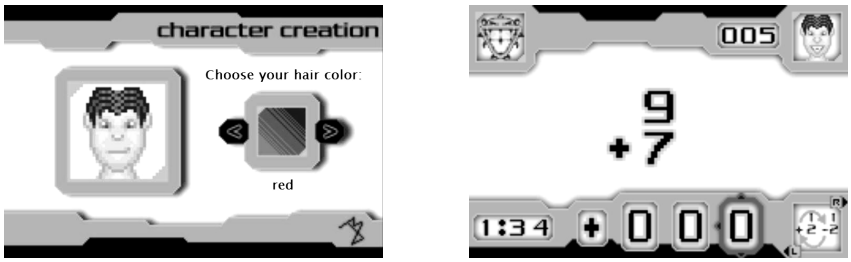
El primero es "Tiro al pato" (figura 3.5a) de Apps (1985). El juego muestra patos a los que hay que disparar (30 en total), pero para poder hacerlo es necesario responder acertadamente a una operación de multiplicación.

La parte de juego (disparar a los patos) es parecida al Space Invaders (figura 3.5b), pero no dispone de ningún tipo de integración con la parte educativa del programa. La motivación es completamente *extrínseca* pues el juego hace las veces de *premio* tal y como dictan las teorías conductistas.

MathWar[31] (figura 3.5c) trata de conseguir que la propia realización de las operaciones sea el juego, añadiendo *competición*. En él, se muestran "cartas"

[30] Los diferentes videojuegos analizados *no* guían el aprendizaje, sino que se limitan a permitir a los estudiantes que practiquen unas habilidades que se supone han sido aprendidas de manera externa.

[31] `http://webpages.charter.net/stuffle/mathwar/MathWar.html`



(a) Creación del perfil (b) Partida

Figura 3.6: Skill Arena en Game Boy Advanced (Lee et al., 2004)

con números sobre los que hay que calcular una operación aritmética sencilla. El resultado hay que darlo antes de que lo haga el oponente, papel jugado por el propio ordenador. Éste proporcionará su solución pasado un cierto tiempo configurable (mostrado en la barra de progreso de la esquina inferior derecha) con una tasa de error también configurable. Se espera que los niños encuentren motivante esta presión por ganar al sistema, dado que es el único ingrediente de videojuego que se incluye.

Comparándolo con el Tiro al Pato, este juego mejora la integración entre la parte lúdica y educativa, haciendo que, de hecho, ambas sean la misma gracias a la competición. Algo parecido hacen Lee et al. (2004) pero esta vez cambiado de plataforma a la consola portátil Game Boy Advanced.

En este caso, permiten crear un perfil personalizado (figura 3.6a) que pretende tener un carácter motivador adicional al *personalizar* al alumno en el propio juego. Además, la presión del tiempo se integra más en el interfaz, pues la operación aritmética *se desplaza* por la pantalla de izquierda a derecha (figura 3.6b), con la intención de crear la sensación de un *objetivo* para aumentar la motivación. Cuando se acierta el valor, se recibe *feedback* inmediatamente, y la operación es sustituida por otra nueva. El objetivo es acertar cuantas más operaciones mejor en los dos minutos que dura una partida. Con el avance del tiempo, los números se desplazan a mayor velocidad por la pantalla, por lo que el alumno debe responder mucho más deprisa. El sistema lleva un registro de las mejores puntuaciones, lo que empuja a la competición entre diferentes alumnos incentivando, quizá, un mayor uso gracias a la *presión social*. Se ha utilizado en una prueba piloto con dos clases de unos 20 alumnos con unos resultados bastante prometedores: el número de ejercicios resueltos por el grupo experimental que usó el juego fue el triple que el del grupo de control.

También el tiempo es el componente usado por "Aprende matemáticas con Pipo". Pertenece a la rama de videojuegos educativos "Aprende con pipo" de Micronet. Es un claro ejemplo de *edutainment*[32], o juegos para niños con

[32]Este término proviene de la mezcla de dos palabras inglesas, *education* (educación) y

(a) Operaciones aritméticas (b) Operaciones de comparación

Figura 3.7: Aprende matemáticas con Pipo (versión de evaluación)

una alta carga de contenido educativo. Se basan en la premisa de que los niños son fácilmente engañables adornando la parte educativa con un aspecto de juego utilizando interfaces infantiles plagados de figuras en movimiento y colores vivos.

En este caso, además de poner en práctica las operaciones matemáticas, también se realizan ejercicios con operadores relacionales, o se hacen preguntas sobre geometría.

En la figura 3.7a se muestra un nivel del juego en el que aparecen varios estandartes con diferentes números que pasan de derecha a izquierda sobre un fondo en el que se ve la Gran Muralla china. El niño debe pulsar sobre el estandarte que contiene el número resultado de la operación matemática mostrada en la parte inferior. Aparte del tipo de interfaz cercano a los niños, como motivación para jugar se encuentra también la lucha contra el tiempo. Cada vez que se acierta una operación matemática se propone otra y hay que averiguar tantas como sea posible en el tiempo que tarda en vaciarse la barra que aparece a la derecha. Ahora no existe competidor más que uno mismo.

En la figura 3.7b el niño debe ordenar de mayor a menor los diferentes números resultantes de las operaciones matemáticas impresas sobre las cestas.

Otro juego educativo que también podemos clasificar como *edutainment* es *Treasure Mountain!*, de *The Learning Company*, una compañía fundada en el lejano 1.980 para desarrollar juegos educativos que, después de pasar por diferentes manos, todavía sobrevive. En este caso la parte de videojuego tiene mucho más peso y la parte educativa aparece más de vez en cuando. Esto hace que la componente lúdica sea más significativa con la intención entretener (¿engañar?) a los niños mucho más.

El cometido del juego es ascender a una montaña recorriendo sus tres niveles mientras se buscan tesoros y las llaves para ascender al nivel superior.

entertainment (entretenimiento), y pretende dar la idea de educación entretenida.

(a) Caza del elfo

(b) Pregunta

(c) Llave encontrada

(d) Comprando un cazamariposas

Figura 3.8: *Treasure Mountain!*

Para ello, es necesario conseguir las pistas sobre su localización, proporcionadas por elfos que deben ser capturados con un caza-mariposas (figura 3.8a).

Naturalmente, los elfos sólo están dispuestos a darnos una pista si somos capaces de responder acertadamente a su pregunta (figura 3.8b), que es donde entra la parte educativa. Las pistas (tres por nivel) son meras palabras (en este caso, *small*, *triangle* y *flower*, parte inferior izquierda de la pantalla). La llave se encuentra en el lugar en el que encajen las tres (figura 3.8c), mientras que los tesoros se encuentran en los lugares donde encajen dos de ellas. Como ingredientes adicionales al diseño se encuentra la posesión de *monedas*, conseguidas al cazar elfos y que son gastadas para comprobar si en algún punto hay un tesoro o una llave, y para comprar nuevos caza-mariposas cuando se nos estropea el que tenemos (figura 3.8d).

Los ejemplos anteriores disponen de lo que Kafai (2001) denomina *integración extrínseca*, dado que la parte educativa aparece de manera un tanto "forzada" en el juego, en lugar de estar integrada de una manera natural en su diseño. La integración extrínseca tiene una ventaja muy clara pues *el dominio es intercambiable*. Podemos reutilizar un determinado diseño de videojuego cambiando el tipo de preguntas sin demasiados problemas (por ejemplo, Tiro al Pato podría hacer preguntas sobre geografía y no habría diferencia). De hecho, *Treasure Mountain!* está enfocado principalmente a potenciar las *habilidades lectoras* de los niños (no a las aritméticas), y al-

gunas de las preguntas que plantea tratan sobre palabras. Pero gracias a la integración extrínseca, *The Learning Company* creó una familia de videojuegos educativos basados en el mismo diseño (la serie *Super Seekers*) al que pertenecía también *Treasure Mathstorm!* que se centra en cuestiones como la lectura de la hora, álgebra e inecuaciones. Del mismo modo, "Aprende matemáticas con Pipo", como se ha dicho, forma parte de una larga serie de videojuegos educativos que, bajo una idea similar, enseñan muchas otras cosas[33] (a leer, inglés, o el cuerpo humano).

Aunque a nivel de *desarrollo* este tipo de integración es beneficioso, puede no serlo tanto para la motivación y el entretenimiento. Lo ideal sería conseguir *integración intrínseca* (que sería la conseguida por la película "Naufrago" en nuestra comparación con las películas y la publicidad) en el que el contenido forma parte *inseparable* del juego. Como último ejemplo de nuestra serie de videojuegos educativos para practicar las operaciones aritméticas hablaremos de "How the west was won" (Burton y Brown, 1976), que consigue esa integración intrínseca. El juego es parecido al juego de tablero de "La escalera", una especie de oca en el que hay numerosas casillas que te hacen saltar hacia delante y hacia atrás. La principal diferencia de "How the west was won" (aparte del tablero concreto y sus "atajos") estriba en que se juega con tres "dados" y el numero de casillas que el jugador se desplazará por el tablero es obtenido mezclando los tres valores aleatorios de los dados con dos operaciones aritméticas elegidas por el propio jugador. En el tablero hay algunas casillas perjudiciales que te hacen retrasar, y otras que te hacen avanzar mucho. Esto supone que no siempre conseguir el número más alto será lo preferible, lo que fuerza a los jugadores a buscar combinaciones aritméticas que les favorezcan según la situación. En este caso, es imposible separar la parte pedagógica de la lúdica (o, al menos, no podemos adaptarlo para que enseñe, pongamos, a leer), lo que ocasiona esa integración intrínseca. No obstante, en este caso estamos hablando en realidad de un videojuego que bien podría jugarse directamente sobre un tablero sin el uso de ordenadores. Burton y Brown lo implementaron digitalmente porque el sistema seguía la pista de combinaciones subóptimas de los jugadores para averiguar si normalmente eludían operaciones complejas como la multiplicación para hacérselo notar.

Conseguir un diseño con *integración intrínseca* no siempre es sencillo. Resulta relativamente obvio que para enseñar lógica booleana podemos hacer un juego en el que haya que construir máquinas o robots que se comporten de un determinado modo (*Rocky's Boots* o *Robot Odyssey*, también de *The Learning Company*); para enseñar planificación urbanística se puede diseñar una ciudad (*Sim City*); o para enseñar la física del movimiento se puede poner al alumno a los mandos de una nave espacial y a disparar misiles (*Lunar lander* y *Asteroids*). Sin embargo, cuando la relación no es tan clara es fácil

[33] `http://www.pipoclub.com`

caer en una integración extrínseca basada en preguntas tal y como ocurre en los ejemplos planteados. Kafai (2001) propone imponer como *condición de diseño* la imposibilidad de *realizar preguntas* en el juego sobre el área, para forzar a los diseñadores a dejar volar la imaginación. Normalmente, esto requiere además un proceso intelectual mucho más arduo pues suele ser necesaria una comprensión mucho más profunda del área que se intenta enseñar. Una alternativa es hacer uso, por ejemplo, de *metáforas* que relacionen el entorno lúdico del juego con el área que se desea enseñar (Gómez Martín et al., 2007b).

Naturalmente, el diseño de un videojuego (educativo o no) es un proceso dinámico que atraviesa muchas etapas según van surgiendo nuevas ideas y se van puliendo o descartando las viejas. Repenning y Lewis (2005) hablan de dos aproximaciones para realizar el diseño de un videojuego educativo, dependiendo de qué extremo se parta (aplicación educativa o juego):

- *Diseño educativo*: en este caso el principal objetivo es el aprendizaje, por lo que se comienza con las ideas relativamente claras sobre la parte educativa, y se va maquillando con elementos lúdicos.
- *Diseño de juego*: se basa en la idea de un videojuego al que se añade carga educativa.

Utilizando la primera aproximación es probable acabar creando *edutainment* en el que la parte educativa es adornada con ingredientes de juego como contenido multimedia, animaciones y mundos virtuales. En el segundo es muy fácil, como ya se ha dicho, caer en una integración a través de interrupciones de preguntas. Lo interesante sería que el aprendizaje que en todo juego se realiza sobre sus reglas y funcionamiento interno se enlazara con el aprendizaje del dominio de una manera fluida.

Llegados a este punto, es necesario reconocer que, al igual que decíamos que no se conocen los ingredientes que hacen a un videojuego *divertido*, tampoco se sabe a ciencia cierta cuales convierten a un juego educativo en uno bueno para enseñar (Kafai, 2001, 2006). De hecho, en general se defiende la integración intrínseca porque los estudios más serios realizados en este sentido encontraron que era la más acertada, pero dado que fueron realizados hace ya bastante tiempo (Malone, 1981b, 1987) estos resultados podrían no ser del todo aplicables hoy en día. Esto significa que, en realidad no se comprenden completamente las mejores maneras de realizar uniones entre las componentes lúdicas y las educativas (Repenning y Lewis, 2005).

3.8. Categorización de los videojuegos educativos

Actualmente el mercado de los videojuegos educativos es relativamente reducido, y está enfocado principalmente hacia el *edutainment* mencionado

previamente. No obstante resulta tentador tratar de categorizar los diferentes productos existentes. De nuevo, al igual que ocurría con los géneros de los videojuegos tradicionales, existen discrepancias y diferentes personas plantean distintas alternativas.

En lugar de decantarnos por una u otra, trataremos aquí de identificar diferentes *ejes* respecto a los cuales organizar la oferta existente. Estos ejes *no* son completamente independientes, y el "espacio" que forman no está, ni mucho menos, poblado de manera uniforme.

El primer punto a considerar en un juego educativo es *¿qué enseña?*. Aquí encontramos principalmente tres alternativas:

- *Contenido lineal*: en la página 19 definíamos contenido lineal como aquel en el que hay un claro *eje de evolución* o *nivel de aprendizaje*. Decimos que un juego enseña este tipo de contenido cuando *guía el aprendizaje*, proponiendo "ejercicios" sencillos al principio que se van complicando de manera paulatina y cuidadosa.

- *Contenido dinámico*: es aquel en el que no es posible ordenar a un grupo de gente de acuerdo a su conocimiento en ese ámbito. Dentro de este tipo de conocimiento están las diferentes cuestiones que se aprendían de manera natural con el uso de videojuegos, como por ejemplo la capacidad de superación o el trabajo en grupo.

- *Principios y valores*: estos juegos entran dentro de lo que suele conocerse como *tecnología de la persuasión* al intentar inculcar un determinado punto de vista en sus destinatarios. Es algo que tradicionalmente han hecho diferentes medios de comunicación (que a menudo tienen un marcado caracter *partidista*), pero también algunas campañas divulgativas e incluso programas de ordenador que persiguen la concienciación ciudadana sobre el medio ambiente, los problemas en el Tercer Mundo, la escasez de agua, etcétera.

La categoría del contenido lineal es un poco difusa. Los juegos descritos anteriormente para practicar aritmética básica seguramente serían, como primera idea, encasillados dentro de este tipo, dado que la capacidad de realizar las operaciones básicas forma parte del *currículo* de primaria, que es contenido lineal. De hecho, resulta relativamente sencillo ordenar varias operaciones matemáticas en función de su *dificultad*, y a los alumnos en función de su *capacidad* para resolverlas.

Sin embargo, esos videojuegos *no* aprovechan ese orden. Plantean ejercicios a discreción sin preocuparse por hacerlo de una manera gradual en lo que se refiere a su dificultad[34]. Ante este panorama resulta cuanto menos

[34] A modo de ejemplo, hemos visto plantear a MathWar la pregunta `12x12` seguida de `0x7`

discutible si encasillarlos o no en esa categoría, dado que *no son tutores* por lo que la "linealidad" del contenido desaparece. Más adelante, especialmente a partir del capítulo 4, hablaremos sobre la forma de incluir esa capacidad pedagógica en los videojuegos, proponiendo un modelo para llevarla a cabo.

Por otro lado, la inculcación de principios y valores podría considerarse contenido dinámico. No obstante, dado su objetivo de concienciación más que de *enseñanza* hemos preferido mantenerlos como tipos diferentes. Ejemplos de esta categoría son *Food Force*[35], que pretende hacer a quienes lo jueguen conscientes de los desafíos logísticos de entregar ayuda alimentaria en zonas que sufren crisis humanitarias. El juego consta de seis fases, donde lo importante no es tanto hacerlo bien como ver la dificultad de estimar el número de afectados, crear paquetes de alimentos básicos económicos pero energéticos o hacerlos llegar por aire o tierra. El último nivel simula la evolución de un poblado y la responsabilidad del jugador es elegir el modo en el que se distribuyen los diferentes recursos, de modo parecido al antiguo e influyente *Hammurabi* (Crawford, 1982, página 82) que Apps (1985) renombraría por "Poblados".

Otro ejemplo es *Darfur is Dying*[36], una simulación narrativa donde el usuario, desde el punto de vista de un refugiado sudaní, debe negociar con las fuerzas que amenazan a su campamento. Pretende concienciar a la ciudadanía (y especialmente a los niños) de las dificultades que más de dos millones y medio de desplazados sufren diariamente en una remota región de Sudán.

El segundo eje de categorización es relativo al *modo de desarrollo*:

- *Juego comercial reaprovechado*: dada la capacidad educativa natural de los videojuegos, es planteable la posibilidad de utilizar juegos comerciales (*off the shelf computer games* en la terminología inglesa habitual) para enseñar. En este bloque se deben situar únicamente aquellos videojuegos que *no* fueron diseñados originalmente con este propósito. Un ejemplo claro es el uso de un videojuego de carreras para el aprendizaje, al menos en una primera aproximación, de los circuitos por parte de pilotos profesionales. En este caso, no se requiere desarrollo de ningún tipo: se coge el juego y se usa.

- *Juego hecho desde cero*: en este caso, se parte de unos objetivos educativos, un planteamiento de diseño, y se comienza la implementación más o menos desde el principio. Aunque sea posible reaprovechar com-

[35] Disponible en `http://www.food-force.com/`, fue creado por el *World Food Programme*.

[36] Fue el ganador de un concurso impulsado por el *International Crisis Group* la fundación Reebok proderechos humanos y la mtvU en el que premiaban modos de concienciar a la ciudadanía sobre el problema de los refugiados de Sudán. Está disponible para jugar on-line en `http://www.darfurisdying.com`

ponentes de juegos realizados previamente o algún "motor" de videojuegos, en este caso la carga de programación durante el desarrollo será bastante alta.

- *Juego comercial adaptado*: dada la cada vez mayor posibilidad de personalizar muchos juegos comerciales (en lo que se conoce como *modding* o, simplemente, *mods*) es posible adaptar el contenido de algunos de ellos para proporcionar algún tipo de enseñanza concreta. Esta categoría se sitúa en el punto medio de los dos anteriores. Se evita así gran parte del desarrollo requerido en el primer caso, pero permitiendo la adecuación del juego a un tipo de enseñanza concreta que podría no tener inicialmente.

Obviamente, en un mundo ideal la mejor opción es la primera: si ya hay un juego comercial que, de manera lateral, se adapta a las necesidades educativas, ¿para qué reinventar la rueda?

Por desgracia, esto ocurre muy raras veces. Es cierto que algunos videojuegos tienen una alta carga educativa, pero antes de ser utilizados en contextos pedagógicos reglados suele ser necesario una adaptación del contenido o, al menos, una preparación previa de los alumnos para guiar el aprendizaje y sacar el mayor partido posible del videojuego. También será necesario convencer tanto a padres como a otros docentes o autoridades de que utilizar un determinado juego comercial no resulta una pérdida de tiempo. Uno de los escasos ejemplos de esta categoría se ha realizado con el juego Civilization III, que recrea la evolución de las civilizaciones desde la Edad del Bronce hasta la conquista espacial. Se simula la consecución de descubrimientos científicos, se muestra la relación entre la creación de infraestructuras, el desarrollo militar o la investigación. Squire y Barab (2004) lo han utilizado con éxito (y mucho esfuerzo) para enseñar las luchas geopolíticas, los vaivenes de la economía global, los detalles geográficos que resultan significativos para el poder regional, y muchos otros conceptos socio-históricos. Increíblemente, la gran motivación del propio juego hacía que los estudiantes recurrieran a los libros de texto tradicionales por propia iniciativa para buscar información que les pudiera ayudar en el desarrollo de sus civilizaciones. Además, se planteaban preguntas del tipo "qué hubiera pasado si...", lo que les llevaba a intentar construir imperios con civilizaciones débiles, enfrentándose a todo tipo de problemas (Squire, 2004).

De todas formas, este caso es muy poco habitual. La otra opción extrema del desarrollo de un videojuego completo puede resultar muy laboriosa. Como hemos dicho, existe la posibilidad de utilizar *motores de juego* que facilitan la tarea en la medida de lo posible, evitando mucho del costoso trabajo de implementación. No obstante, incluso en ese caso es necesario un número bastante alto de habilidades dentro del grupo de desarrollo (Repenning y Lewis, 2005). Esta necesidad de conocimiento y tiempo, supone que la

mayoría de las veces el resultado esté lejos de los videojuegos comerciales lo que repercute en el interés de los estudiantes al no cumplir sus expectativas. En este sentido, Elliott et al. (2002) nos cuentan que, en una prueba piloto con un prototipo para enseñar matemáticas llamado AquaMOOSE, al ser un alumno preguntado por la fase en la que se practicaban las coordenadas polares, éste respondió "no recuerdo nada salvo el pez pequeñajo y horrible".

En realidad, muchas veces los videojuegos educativos no necesitan la cantidad de sofisticación de los videojuegos comerciales, lo que puede permitir el uso de *middleware* utilizado en el conocido como *mercado independiente* en el que los requisitos no son tan exigentes y pueden ser desarrollados por un equipo más reducido y en menos tiempo. Por ejemplo, Dimenxian[37], un videojuego educativo comercial para enseñar geometría, está desarrollado con Torque[38], un motor pensado para el desarrollo de videojuegos de bajo presupuesto.

Otra alternativa es crear videojuegos aún más sencillos utilizando herramientas pensadas para desarrollo rápido y corto como Macromedia Flash o AgentSheets. En este caso el tiempo se desplaza del desarrollo hacia el diseño, como solía ocurrir en los videojuegos comerciales de hace un par de décadas donde lo importante es la jugabilidad y originalidad en lugar de los alardes tecnológicos. Repenning y Lewis (2005) denominan a este tipo de sistemas *gamelets*.

En cualquier caso, si se quieren desarrollar videojuegos educativos con un aspecto similar al de los videojuegos comerciales, una alternativa razonable es la última presentada, en la que se reutiliza un juego comercial completo modificando o creando completamente el contenido (niveles de juego). Esto permite aprovechar lo último en tecnología con el mínimo esfuerzo de desarrollo. La parte negativa es que se condiciona fuertemente el diseño, que debe poder ajustarse a las reglas del videojuego original. Un ejemplo de esta aproximación es Revolution (Squire y Jenkins, 2003) en el que el aprendizaje se realiza mediante un juego de rol a través del que el estudiante participa en la vida de una ciudad colonial norteamericana durante la revolución. Cada uno tiene un papel en la colonia, con sus responsabilidades diarias y con sus opiniones políticas (leales, revolucionarias o neutrales). El sistema está creado sobre el conocido juego de rol Nevewinter Nights. Otro ejemplo dentro de esta categoría es el trabajo realizado por Purushotma (2005) en el que mezcló el contenido de las versiones inglesas y alemanas del conocido juego *Los Sims* para enseñar vocabulario de alemán.

A menudo, sin embargo, la decisión entre el desarrollo desde el principio o la modificación de un videojuego existente depende principalmente de los conocimientos del equipo que desarrollará el videojuego y de sus objetivos. Esto tiene bastante relación con el siguiente eje de categorización, que se

[37] `http://www.dimenxian.com/`
[38] `http://www.garagegames.com/`

refiere a las *razones para la creación* del videojuego:

- *Razones comerciales*: en este caso el objetivo último del desarrollo es *su venta* o explotación comercial de alguna forma. Pueden buscar un mercado generalista para intentar llegar al mayor número de gente o bien ser desarrollos a medida para situaciones específicas. La mayor parte de las veces estaremos en el primer caso. El mercado del *edutainment* es relativamente amplio y entra dentro de esta categoría. Los videojuegos educativos *a medida* son aún escasos debido a la poca concienciación de los centros de formación de todo su potencial.

- *Investigación*: existe un creciente interés en el mundo académico por el uso de videojuegos para la enseñanza. Para poder poner a prueba diferentes ideas diversos grupos han desarrollado prototipos específicos adaptados para analizar sus ventajas e inconvenientes con grupos de usuarios experimentales, o para plantear ideas en la integración entre la parte lúdica y educativa.

- *Uso interno*: a nivel particular, un profesor con inquietudes y conocimientos suficientes podría desarrollar o adaptar un videojuego para utilizarlo en clase. Dado el interés que están levantando las TIC en el ámbito educativo, algunas instituciones públicas están intentando impulsar usos nuevos proporcionando cursos de formación que bien podrían ir en esta linea. No obstante, dado que los juegos creados de esta manera parten de iniciativas individuales, es difícil conocer su penetración.

Los desarrollos comerciales se encuentran en una situación parecida a los estudios de desarrollo de videojuegos: en función del tipo de juego, dinero disponible y tiempo se decantarán por crear el juego completo, apoyarse en algún motor de juego o por desarrollar un *mod*. En los tres casos es necesario vigilar *las licencias* de las herramientas que utilicen.

Cuando el uso es para investigación o a nivel particular por parte de un profesor, como no existe un *ánimo de lucro* normalmente no hay problema con dichas licencias. Eso empuja a muchos grupos a realizar modificaciones de juegos existentes por el tiempo ahorrado y por evitar tener que comprender completamente la tecnología que subyace a los videojuegos. En algunos casos, no obstante, cuando el videojuego es relativamente sencillo o tiene unos requisitos muy específicos difícilmente encontrados en juegos comerciales que puedan ser aprovechados, la alternativa de la creación completa adquiere más peso. Por último, también hay algunos usos de videojuegos comerciales sin modificación, aunque resultan aún relativamente tímidos.

Como decisión de diseño a menudo es necesario elegir el *uso principal que se quiere dar al videojuego*, lo que constituye un nuevo eje de categorización.

La mayor parte de las veces se realizan juegos pensados para *repasar*, de modo que se aprovecha la capacidad de motivación para conseguir que los estudiantes hagan más ejercicios de los que harían en otro caso. Se mantiene así el aprendizaje conductista por prueba y error (Squire, 2003). Aprovechando la evolución del *lenguaje* de los videojuegos, que permiten una gran cantidad de alternativas para enriquecer los entornos (González Calero et al., 2006), hoy es posible un aprendizaje por descubrimiento sobre el que posteriormente realizar un análisis crítico junto con el tutor con el que aclarar las cosas o explicar los cabos sueltos.

Una alternativa mucho menos habitual es que el propio juego guíe el aprendizaje y no se requiera un agente externo que lo apoye. En realidad, dado que la mayor parte del aprendizaje en los videojuegos comerciales es "emergente", es una pena que este ámbito se haya explorado tan poco.

Independientemente de si sirven para repasar o para enseñar (de manera autónoma o no), será necesario tomar medidas concretas si se aspira a que el videojuego sea utilizado directamente en una clase reglada. En breve exploraremos este aspecto con algo más de detalle.

Por último, hay que considerar *el género* del propio videojuego (sección 3.3.1). Algunos géneros parecen más adecuados que otros para la docencia, siendo *los simuladores* los más claramente aprovechables en la educación (no es de extrañar, dado que también aparecieron en el capítulo 2 como herramientas de enseñanza).

En las siguientes subsecciones entraremos un poco más en profundidad en algunos de los aspectos mencionados aquí que requieren una atención especial.

3.8.1. Dificultades para el uso en clase

Aunque Bork se aventuraba en 1981 a decir que en el año 2.000 la mayor parte del aprendizaje sería interactivo y basado en ordenador, el tiempo ha demostrado una vez más que los visionarios no siempre aciertan.

Consideramos que hay tres dificultades principales por las que los videojuegos educativos no han entrado aún masivamente en las aulas:

- Preparación de los profesores: no está aún completamente superada la barrera de los propios *profesores*. Para que una determinada tecnología se utilice, los docentes no sólo tienen que estar convencidos de sus bonanzas, sino que deben *entenderla*.

- Dado que el uso de videojuegos requiere *ordenadores* es necesario disponer de ellos en los centros educativos o, mejor aún, en las propias aulas.

- Los videojuegos, incluso los educativos, tienen que estar pensados para ser usados en clase. Aspectos como la duración de cada sesión de uso,

> o la cantidad de sonido que emiten son determinantes para que los profesores les abran las puertas de sus aulas.

La falta de conocimiento en *informática aplicada* por parte de un gran número (afortunadamente decreciente) de profesores supone una primera barrera para conseguir el aprovechamiento de las TIC en general y de los videojuegos en particular dentro del aula. La administración es muy consciente de este problema por lo que dedica parte de sus fondos en formación al profesorado a este aspecto. Estos conocimientos deben cubrir en principio aspectos básicos sobre informática para perder el miedo al uso de ordenadores a nivel de usuario y poder así comenzar a integrarlos como herramientas habituales para impartir sus clases.

El siguiente paso debe potenciar el uso de ordenadores de un modo más sofisticado, buscando usos más evolucionados o novedosos. Es de esperar que el *tiempo* juegue a favor de este aspecto, según vayan entrando profesores jóvenes que han nacido en la era digital y encuentran en éstos un modo natural de comunicación y expresión.

En cualquier caso, resulta de interés la creación de redes de colaboración para aunar esfuerzos y compartir experiencias, de forma que las ideas individuales de los profesores con más inquietudes salgan más allá de las puertas de su propia aula. En este sentido, un esfuerzo muy interesante a nivel de evaluación de software educativo (no sólo de juegos) para el uso en clase es el realizado por el grupo TEEM[39], que analiza gran cantidad de software (el último catálogo detalla unos 800 productos) en función del modo en el que puede utilizarse en clase. Por desgracia, no tenemos conocimiento de ninguna iniciativa similar para software en castellano.

El empeño de los profesores por utilizar las TIC no resulta útil si los centros educativos no disponen de ordenadores que puedan utilizarse. Las estadísticas publicadas por el MEC (Ministerio de Educación y Ciencia) indican que en España durante el curso 2004/2005 cada ordenador destinado al uso por parte de los alumnos debía ser compartido por 9 ó 10 de ellos. Si se particulariza este dato únicamente en los centros de Educación Secundaria, encontramos que cada ordenador es compartido por 7 alumnos. Aunque esta cifra parece alta, es prometedor el descenso que se ha venido apreciando en este dato en los últimos años; en el curso 2002/2003 había 16 alumnos por cada ordenador.

Naturalmente, esta cifra no es la misma en todas las Comunidades Autónomas. Destaca muy positivamente Extremadura, en el que hay un ordenador para cada sólo 3 alumnos; en esta situación tiene mucho que ver el impulso de *Linex* (la primera distribución de GNU/Linux creada por un gobierno aunque sea regional) que esta Comunidad Autónoma viene proporcionando desde hace ya varios años.

[39] `http://www.teem.org.uk/`

Otro potencial problema a considerar es el tipo de ordenadores y el sistema operativo subyacente. Aunque varía mucho, algunos videojuegos educativos, especialmente los creados a partir de motores de juegos comerciales, pueden ser muy exigentes en requisitos de *hardware*. Las estadísticas anteriores se limitan a hablar del número de ordenadores disponible, pero no hablan sobre su calidad. El descenso de la ratio alumno/ordenador se debe a la adquisición de nuevos ordenadores, pero los anteriores *siguen en uso*. Eso significa que el parque de ordenadores en los centros escolares puede tener una antigüedad media de varios años, imponiendo problemas en el uso de software algo más moderno.

Asumiendo superadas las barreras del conocimiento del profesorado y de la disponibilidad de ordenadores, el siguiente paso es analizar la viabilidad del uso de software educativo en las propias clases.

Deubel (2002) sugiere que antes de utilizar un determinado software educativo los profesores deberían estar seguros de que permite ser modificado para adaptarlo a cada caso particular, que está correctamente alineado con el currículo y que realiza un seguimiento del progreso de los alumnos.

McFarlane et al. (2002) analizó la posibilidad del uso de videojuegos en clase, tanto productos pertenecientes a la categoría de *edutainment* como juegos comerciales tradicionales (*Formula One Racing*, *Age of Empires*, o *Los Sims*). Echan de menos los siguientes aspectos:

- *Exactitud de los contenidos*: en los títulos que contienen simulaciones, normalmente aparecen respuestas y reacciones que no se producirían en la realidad, lo que estropea el aprendizaje correcto. En los títulos en los que esto no ocurre, los prejuicios de los profesores hacen que no se confíe en ellos, por lo que debería ser dicho explícitamente que se basan totalmente en la realidad.

- *Posibilidad de guardar y cargar*: aunque a nivel de diseño de juego puede resultar interesante permitir únicamente grabar y cargar en puntos concretos, debido a las fuertes restricciones de tiempo en clase resulta importante poder salvar la partida en el estado exacto en el que se encuentre al llegar la hora de salida. Esto evita perder tiempo al día siguiente para llegar a esa situación desde el último punto guardado. Por otro lado, como habitualmente se trabaja en grupo, la tendencia habitual a asociar cada perfil a una única persona resulta incómoda. Por último, en un aula los alumnos no necesariamente tienen asignados puestos fijos, pues los ordenadores podrían moverse de aula, estropearse, o estar ocupados para otras tareas de manera bastante dinámica. Por tanto, resultaría interesante que el avance de cada perfil se mantuviera en algún sitio centralizado para evitar la dependencia de un grupo de alumnos con un ordenador concreto.

- *El papel del sonido*: cuando hay muchos ordenadores simultáneamente

en una clase resulta contraproducente que se reproduzcan sonidos en cada uno de ellos, pues la música de fondo o efectos sonoros de un ordenador molestarán al resto de alumnos. La alternativa de utilizar cascos no es práctica si pensamos en que la mayoría de las veces el uso de un ordenador es compartido. Incluso en el caso de que pudieran conectarse varios auriculares al mismo ordenador para que cada alumno del grupo pudiera escuchar los sonidos, su uso limita la posibilidad de que los propios estudiantes hablen entre sí. Esto dificulta la cooperación entre ellos, perdiéndose uno de los aspectos educativos transversales importantes del uso de videojuegos.

- *Información al profesorado*: las editoriales de libros educativos y cuadernos de prácticas suelen proporcionar guías al profesor con consejos sobre el mejor modo de utilizar el material. Pocos títulos de *edutainment* disponen de algo equivalente, y mucho menos los juegos comerciales que, en ocasiones, pueden usarse en un aula.

Rice (2005a) propone una serie de preguntas que deberían ser contestadas antes de decantarse por el uso de un determinado videojuego:

- *¿El juego se adapta al currículo?*: normalmente esto no ocurre, y ajustar el contenido, cuando es posible, puede llegar a suponer mucho trabajo. Aquí de nuevo aparece la importancia de las redes de colaboración, para que el esfuerzo de adaptación realizado por un determinado profesor pueda ser reutilizado en contextos similares por otros.

  En realidad, como hemos visto, el uso de videojuegos es interesante directamente por la gran cantidad de habilidades que ponen en práctica; sin embargo muchos padres aún no son conscientes de esto y para un profesor resulta mucho más difícil argumentar el uso de videojuegos que no encajan completamente con alguna parte del currículo.

- *¿Es el juego fácilmente personalizable?*: tiene relación con el punto anterior para conseguir adaptar un juego a la situación particular de cada clase. Pero en este caso el interés es más bien sobre el coste de esa personalización en tiempo. La situación en el aula es muy cambiante y una pregunta de un alumno podría hacer surgir la idea de un nivel nuevo en el juego o un ajuste fino de alguno ya existente para ejemplificar la respuesta del profesor. Si la personalización es laboriosa o requiere herramientas complicadas que sólo pueden utilizarse de manera reposada se pierden oportunidades de aprendizaje. Además, dependiendo de su facilidad de uso, las herramientas de adaptación podrían también ser usadas por los propios alumnos para experimentar, abriendo un campo nuevo para el aprendizaje.

- *¿Sirve el juego para practicar o para enseñar?*: esto tiene relación con los simuladores, dado que la separación entre un simulador y un juego

es muy fina (y subjetiva). Dependiendo de la situación, se preferirán juegos que sirvan para practicar, de modo que los alumnos realicen más ejercicios de los que harían en otro caso, y otras veces se preferirán sistemas que *enseñen*, sustituyendo parcialmente al profesor que pasará a tomar un papel de guía y consejero más que de fuente de conocimiento.

Por último, es necesario hacer notar que dado que los videojuegos tienden a buscar el aprendizaje por descubrimiento, suelen requerir más tiempo que la exposición por parte del profesor del mismo contenido que el software intenta enseñar. Esto no necesariamente es una desventaja. En principio, el aprendizaje participativo y activo es mucho más efectivo tal y como, por ejemplo, nos dice el cono de aprendizaje de Edgar Dale[40]. Pero, además, el rendimiento de un alumno se ve incrementado no sólo por sus habilidades, sino también por su *motivación* (Long, 2006), por lo que los juegos, gracias a su carácter motivador y activo, consiguen la mejor parte de ambas cosas. Que su uso requiera más tiempo no necesariamente es un problema si la diversión que provocan hace que los estudiantes dediquen con gusto ese tiempo a su uso en lugar de dedicarlo a otras cosas. No obstante, podría significar que el uso de videojuegos sea más adecuado para el uso extraescolar que en clase (Rice, 2005b).

3.8.2. Los géneros

Los comentarios anteriores que se encuentran en realidad enfocados a los profesores como consejos para la elección de software educativo, también pueden verse como solicitudes a los diseñadores de dicho software para que las aplicaciones que desarrollen se adapten mejor a lo que los docentes demandan y necesitan.

Esto es debido a que durante la fase de diseño es necesario mantener en mente el *público objetivo*. En este sentido, también conviene plantearse a qué *género* (sección 3.3.1) pertenecerá el juego final desarrollado. Naturalmente, algunos géneros encajan mejor con unos dominios que otros, pero cuando existe cierta posibilidad de elección resulta interesante analizar lo que los usuarios finales preferirán. Esto fue lo que llevó a Amory et al. (1999) a analizar los gustos de los *estudiantes de la carrera universitaria de Biología* como paso previo a la realización de videojuegos educativos de ese área de conocimiento. Aunque el estudio no es demasiado profundo, dedujeron que preferían los juegos de aventura, luego los de estrategia, de acción y, en último lugar y muy alejado, los de simulación.

La penalización a este último género la justifican por su mayor dificultad tanto en las reglas como en el propio interfaz (el juego que utilizaron para el

[40]Edgar Dale publicó en 1.969 un modelo sobre los modos de aprendizaje, según el cual a las dos semanas se recuerda un 10 % de lo leído, un 20 % de lo escuchado, un 30 % de lo visto, un 50 % de lo visto *y* oído, un 70 % de lo *dicho* y un 90 % de lo *dicho y hecho*.

estudio fue *SimIsle*, de Maxis), haciéndolos poco adecuados para gente que no juega mucho. En cualquier caso, resulta un tanto preocupante, pues en muchos dominios es muy natural crear un videojuego educativo que sea un simulador.

De hecho, según Quinn (1994) la mayor parte de los videojuegos educativos son o bien aventura o bien simulación. Aunque en realidad también hay algunos videojuegos educativos de acción significativos como "The Monkey Wrench Conspiracy" (Prensky, 2001) sí es cierto que, como ya vimos, la simulación tiene un papel muy destacado en la educación. No obstante, para muchos la separación entre un juego y un simulador es tan difusa que se consideran una única cosa. Siemer y Angelides (1995) llama "simulación jugable" a cualquier ejercicio de toma de decisiones secuencial, en el que se proporciona un entorno artificial pero realista donde experimentar inmediatamente las consecuencias de las acciones. Esto permite conseguir una comprensión profunda del sistema potencialmente complejo subyacente, así como desarrollar habilidades en su manejo.

Si no se hace una lectura sesgada de esta definición teniendo en mente un tipo de sistema específico, es fácil darse cuenta de que encaja tanto en los simuladores (de aviones u otros vehículos, centrales nucleares, circuitos eléctricos), como en los videojuegos. De hecho, Aldrich (2005) encasilla como juegos de simulación prácticamente a cualquier cosa. Por ejemplo, en los juegos que nosotros hemos categorizado como *aventura*, dice, se "simula" un mundo, unos personajes y una historia (aunque a nivel de implementación esté "cableado" y no sea una simulación totalmente libre).

Un modo de justificar esta visión tan extremista es crear la idea de simulador *inmersivo* y *no inmersivo*. Para Warren Spector[41], un simulador inmersivo es aquel en el que se simula *un mundo completo y sus reglas*. Los simuladores no inmersivos se limitan a *experiencias concretas* dentro del mundo que recrean, es decir a "puzzles" de una única solución. En cualquier caso, quizá la única forma de separar un simulador de un juego sea ateniéndonos a su "diseño". Según Taylor y Walford (1978), en realidad ambos muestran entornos que facilitan el aprendizaje o la adquisición de habilidades; pero los simuladores lo hacen imitando la realidad, y los juegos proporcionando retos motivantes.

También a nivel de implementación hay diferencias. Dependiendo de para qué se quiera utilizar el simulador, querremos que sea más o menos preciso. En aplicaciones científicas y críticas, de hecho, querremos que sea lo que se conoce como *simulador predictivo* es decir, basado en un modelo completamente exacto para poder predecir sin género de duda lo que ocurrirá ante un cambio.

En un videojuego esto no suele ser posible por las restricciones de tiempo real. Del mismo modo, en muchos sistemas educativos se desea enseñar un

[41] Según Aldrich (2005), en una entrevista personal.

área para la que resulta *imposible* crear uno de tales simuladores, como por ejemplo la evolución económica, o la predicción meteorológica. En ambos casos se utilizan *simuladores descriptivos* que se basan en un modelo que *se aproxima* al comportamiento global.

Will Wright[42] pone un ejemplo muy representativo. Si quisiéramos hacer un simulador predictivo sobre el resultado de colocar una pequeña bola sobre la punta de un cono, el modelo analizaría la configuración hasta el último detalle para decidir por qué parte del cono caerá la bola. Un simulador descriptivo utilizará una variable aleatoria. Naturalmente, no será capaz de averiguar el resultado correcto, pero obviamente nos sirve para enseñar sólo combinaciones válidas (la bola nunca irá hacia arriba) y mostrará que uno no puede, después de todo, averiguar el resultado o defenderse de la aleatoriedad.

3.8.3. Un tipo concreto: el edutainment

Ya hemos comentado que el *edutainment* es el término utilizado habitualmente para referirse a la mezcla de educación y entretenimiento. Hoy en día normalmente se asocia a los videojuegos educativos para niños, aunque el término es más antiguo y engloba también a otros medios, como los programas de televisión del estilo de Barrio Sésamo o Los Lunnis.

Hay un mercado relativamente amplio de *edutainment* porque en general los padres están interesados en acercar la informática a sus hijos desde edades tempranas, y buscan un modo divertido y educativo de conseguirlo. También se debe a que resulta mucho más fácil conseguir que un niño se entretenga a la vez que aprende. Normalmente se basan en *contenido crudo* que es adornado con dibujos de personajes infantiles, animaciones y sonidos. Desde el punto de vista de un adulto, esto sigue siendo *contenido crudo*, pero los niños quedan absortos al ver a sus personajes hablándoles y moviéndose para ellos y disfrutan a la vez que aprenden.

Esta fórmula de unión entre entretenimiento y diversión que se aplica en el *edutainment*, sencillamente, no funciona para los adultos. La fusión de contenido, imágenes, animaciones y sonido no se llama juego, se llama sistema multimedia. Naturalmente, este tipo de sistemas proporcionan una buena fuente de conocimiento, pero es discutible que resulte divertido. Según Kopec y Thompson (1992, página 94) si se compara con el aprendizaje tradicional a través de libros, el mero hecho de utilizar ordenadores proporciona diversión. En realidad, la referencia es algo antigua. No se pretende poner en tela de juicio aquí si un buen contenido multimedia es mejor o peor que un libro de texto. Lo que sí es dudoso es que *hoy* el mero hecho de utilizar un ordenador sea motivante. En 1992 la penetración de la informática era relativamente escasa, y sus posibilidades visuales y sonoras aún no habían

[42] También según una entrevista que aparece en el mismo libro. Will Wright es el creador de, entre otros juegos de éxito, Los Sims.

llegado al gran público. Por tanto, cerrar un libro y utilizar un ordenador resultaba toda una novedad: la tecnología era motivante por sí misma. Hoy en día, para la mayor parte de la gente ponerse delante de un monitor y un ratón no es algo fuera de lo normal, por lo que la parte motivadora del uso de un ordenador ya no existe. Esto exige un trabajo adicional para desarrollar sistemas atractivos que potencien su uso. Naturalmente, utilizar un buen programa multimedia seguramente siga siendo más entretenido que leer un libro, pero no lo suficiente como para dedicar horas y horas a él, algo que *sí* continúa ocurriendo con los buenos juegos.

En cualquier caso, que la mezcla entre parte lúdica y educativa de los productos catalogados como *edutainment* no funcione en los adultos no es importante, dado que estos productos no se enfocan a ellos. Los niños disfrutan con este tipo de juegos y les sirven, como ya se comentó, como un primer acercamiento a los ordenadores, algo que les resultará valioso en el futuro.

La oferta de *edutainment* es relativamente amplia, y ya hemos mencionado algunos ejemplos. Habitualmente son, en realidad, *series* de software, en el que se aprovecha un grupo de personajes y diseños para diferentes áreas y/o niveles. Así lo hacen, en español, la serie de Pipo, de Paula o de los Lunnis de Micronet[43], la serie Abby, Aymun o Gartu de CMY Multimedia[44] (distribuídos por Sirium Entertainment[45]) o la serie Mia de Play & Learn[46].

En lengua inglesa las series más representativas son las de The Learning Company[47] ya mencionada. Junto con la serie *Super Seekers* encontramos la serie *Reader Rabbit*, *Super Solvers* y la destacada *Carmen Sandiego*, originalmente creada por Brøderbund Software. También es conocida la serie *Star Wars* por ser creados por *Lucas Learning* una organización fundada por el mismo George Lucas para el desarrollo de software educativo divertido.

En el lado del software libre, destacan GCompris[48], Childsplay[49] y KDE-Edu[50], que son colecciones de software educativo para niños, incluyendo algunos juegos. Este tipo de software ha llevado a la creación de distribuciones de Linux adaptadas a los centros de enseñanza, ya sean generalistas como DebianEdu o Edubuntu, o creadas por gobiernos y Comunidades Autónomas como *Linex*, *Max* o *mEDUXa*.

En lo que se refiere al modo de uso de este tipo de software, Pérez Martín y Ruíz (2006) encontraron que el 53'8 % de los menores de 6 años juegan mano a mano con amigos y familiares varias veces a la semana. Sólo el 38'46 % juegan con el sistema sin ninguna compañía. No obstante, es importante

[43] `http://www.micronet.es`
[44] `http://www.cmymultimedia.com`
[45] `http://www.sirium.es`
[46] `http://www.playlearnmultimedia.com/`
[47] `http://www.learningcompany.com/`
[48] `http://gcompris.net/`
[49] `http://childsplay.sourceforge.net/`
[50] `http://edu.kde.org/`

mantener un *seguimiento* en la forma de ocio de los críos, por parte de los padres mientras juegan en casa, y de las instituciones escolares si en ellas se utilizan programas educativos. Lo que no está tan clara es la opinión de los propios niños. Gee (2004) cuenta que su hijo, habitual consumidor de este tipo de juegos, le pidió explícitamente que no jugara con él, porque prefería jugar solo.

3.9. Problemas de los videojuegos educativos

Las secciones anteriores han descrito las bondades de los videojuegos educativos, pero, naturalmente, no son la solución a todos los problemas. La principal ventaja de los videojuegos es que *aumentan la motivación*, lo que impulsa a un mayor uso del sistema educativo, mejorando, idealmente el aprendizaje. Por desgracia, el tipo de enseñanza insertada en un juego, al estar basada en el descubrimiento, puede resultar más lenta (aunque, según las teorías más recientes, esto también ocasiona un aprendizaje más profundo). Además, el tiempo también puede verse incrementado debido a los objetivos adicionales (más allá del propio aprendizaje) que son los realmente divertidos pero que también hay que conseguir.

El resultado es que por un lado el aprendizaje se alarga, pero por otro se consigue que los alumnos le dediquen más tiempo. El objetivo es que el incremento en la duración se vea compensado por la motivación: es decir hacer el aprendizaje el doble de largo, pero diez veces más divertido.

Esta justificación sólo tiene sentido bajo dos condiciones: no hay restricciones de tiempo, y hay problemas de motivación (con el potencial abandono correspondiente). En otro caso, las ventajas del uso de videojuegos se debilitan. Al fin y al cabo, todos los juegos tienen *ineficiencias* en sus reglas: hay métodos mucho más eficientes para meter una pequeña bola blanca en 18 agujeros que el propuesto por las reglas del golf, pero no son tan divertidos (Aldrich, 2005, página 85). Por tanto, es necesario analizar el público objetivo para comprender sus circunstancias y decidir cuantos elementos de juego añadir al sistema, sabiendo que un incremento en este aspecto supondrá más diversión, pero también más tiempo.

Esta necesidad de mayor dedicación se añade a las críticas vertidas sobre los videojuegos por aquellos que mantienen que la educación debe ser algo serio. Como ya vimos en la sección 3.8.1, todo esto impone mucha inercia negativa en la incorporación de los videojuegos en los centros educativos.

Pero no todas las críticas se basan en ese tipo de prejuicios, y de vez en cuando aparecen voces con opiniones negativas pero bien fundamentadas. Siemer y Angelides (1995), por ejemplo, critican los videojuegos educativos porque siguen el concepto de aprendizaje por descubrimiento. Aparte del incremento en el tiempo necesario ya comentado, esto significa que el propio juego *no* necesariamente persigue una enseñanza concreta. Al fin y al ca-

bo, los alumnos aprenden por "rendimiento", en lugar de mediante "diálogo socrático", por lo que a menudo no adquieren la habilidad conceptual para comprender los procesos y procedimientos que hay por debajo. Un razonamiento parecido lleva a Quinn (1994) a asegurar que los juegos tienen un soporte pedagógico insuficiente.

En realidad, el principal problema es que la mayoría de los videojuegos *no evalúan ni se adaptan* al estudiante. Podemos compararlos con los simuladores de la sección 2.8: proporcionan aprendizaje pero necesitan un guía que proponga ejercicios y realice el análisis posterior del uso del sistema. Es decir, un *simulador* no es suficiente; para que realmente sea útil hay que incluir métodos instruccionales adecuados. De Jong y van Joolingen (1998) identificaron tres tipos de medidas:

- Proporcionar información sobre el dominio que se enseña, y presentarla de manera simultánea a la simulación cuando resulte necesaria.
- Proponer tareas y retos en los momentos oportunos.
- Si el dominio es complejo, seguir una aproximación en el que la dificultad vaya incrementándose de manera gradual para proporcionar un ritmo de aprendizaje más adecuado.

Resulta curioso que normalmente los *videojuegos comerciales* bien diseñados *no padezcan* estos problemas. De manera intuitiva, los buenos diseñadores son capaces de añadir de manera natural estas tres medidas en los tutoriales iniciales o en el modo de hacer evolucionar la dificultad de los niveles.

Lo ideal sería conseguir que los videojuegos educativos estuvieran igual de bien diseñados para tampoco sufrir estos mismos problemas. Sin embargo, no sólo se trata de que los usuarios superen los diferentes niveles, sino también que *aprendan* la materia que se quiere enseñar. Para eso, es necesario seguir la pista sobre los problemas que está teniendo el estudiante, para tomar las medidas adecuadas. Así, será necesario mostrar información de ayuda en los momentos oportunos, pero sin hacerlo de una manera intrusiva que haga perder la sensación de inmersión, ni tampoco de forma reiterada que rompa la libertad y por lo tanto el aprendizaje por descubrimiento.

Por otro lado, para poder proponer tareas de una manera gradual es necesario disponer de ellas. El orden puede estar cableado y decidido de antemano por parte del diseñador, o ser elegido por el propio sistema. Esto es sospechosamente parecido a la diferencia entre los sistemas basados en marcos y los tutores inteligentes del capítulo 2. Tendríamos así dos alternativas claras:

- *Contenido cableado*: la parte de tutoría se aproxima más a los sistemas basados en marcos. En el lado lúdico, estaríamos ante un sistema en el

que los niveles son creados manualmente a través de herramientas que los diseñadores utilizan para especificarlos completamente. Esta es la aproximación más habitual, tanto en los videojuegos comerciales como en los educativos (aunque en estos últimos, ni siquiera suele haber una preocupación por conseguir una dificultad creciente).

- *Contenido procedimental*: en este caso nos acercaríamos hacia los tutores inteligentes al disponer de *conocimiento* para el análisis y la toma de decisiones, superando la *implementación de decisiones* del modelo anterior. En el caso de los videojuegos, estaríamos en sistemas de "final abierto" en el que cada partida puede evolucionar de una manera relativamente arbitraria. En el mundo de los videojuegos comerciales esto suele ocurrir en los simuladores; en el campo de los videojuegos educativos esta aproximación no se utiliza fuera de un número muy reducido de grupos de investigación.

El principal problema al intentar meter ese conocimiento de tutoría es su *dificultad*. El desarrollo de sistemas inteligentes de tutoría tradicionales es ya de por sí muy complejo y dependiente del dominio. Si queremos añadir componentes de juego aparecen complicaciones adicionales, por la necesidad de conocimientos sobre la tecnología subyacente. La alternativa que planteábamos para este problema era modificar videojuegos comerciales para añadir nuestro contenido. El problema es que superponer a un videojuego tradicional un ITS requiere un esfuerzo de integración y desarrollo mucho más complejo que la mera creación de contenido, por lo que a menudo esta alternativa deja de ser posible. Esto significa que es necesario crear un videojuego desde el principio (quizá aprovechando motores o armazones que alivien un poco el trabajo) lo que exige conocimientos detallados en muchas áreas. A éstas hay que incluir además la formación necesaria para crear la parte de ITS, habilidad en diseño de videojuegos para conseguir integración intrínseca, y lograr, además, enlazar la parte de ITS enfocada al aprendizaje óptimo con la parte de diversión y juego que persigue objetivos totalmente opuestos.

Conclusiones

Comenzábamos el capítulo mencionando la increíble importancia que tiene el juego en el desarrollo motriz e intelectual del niño. Los juegos crean un marco seguro donde practicar el tipo de habilidades que necesitarán en el futuro.

Los *videojuegos*, se convierten, del mismo modo, en una ventana al mundo de los ordenadores, a través de la cual los jóvenes se introducen en un mundo digital. Muchos adultos han tenido que entrar en ese mundo a regañadientes, pasando a incrementar el número de los *inmigrantes digitales*. El

salto generacional causado por este hecho es inmenso. Los jóvenes de hoy tienen habilidades e intereses diferentes a los que tuvieron sus antecesores a la misma edad: no son más despistados o más torpes; sencillamente son buenos haciendo otras cosas. También el modo de *aprender* y de enfrentarse a los problemas es diferente. Esto deja obsoleto el modelo de educación tradicional que venimos arrastrando durante tanto tiempo. El contexto ha cambiado y, sencillamente, apenas funciona.

La pregunta que asalta ante este panorama es si sería posible canalizar todo el interés mostrado por gran parte de adolescentes por los videojuegos para que *aprendan* a la vez que se divierten. En realidad este aprendizaje ya se produce (pues es necesario *aprender* cómo funciona un juego para superarlo), por lo que no resulta tan descabellada la idea de intentar añadir enseñanza *intencionada* dentro de un videojuego. De hecho, casi desde que nació el área de los videojuegos se pensó en esta posibilidad, aunque no es hasta hace relativamente poco tiempo cuando se le está empezando a dar más importancia.

A lo largo de este capítulo, hemos descrito algunos ejemplos de este tipo de videojuegos, así como los problemas de *integración* entre la parte lúdica y la educativa para conseguir una unión perfecta entre ambas. Por desgracia, no todas las noticias son positivas. El principal problema de la mayoría de los videojuegos estriba en que *no guían el aprendizaje*. La parte educativa se realiza de manera manual, cableando, *como mucho*, las decisiones tomadas por los diseñadores sobre el mejor orden de presentar el material. La investigación en el campo de la tutoría inteligente no ha entrado aún en el mundo de los videojuegos (ni de los productos multimedia en general), por lo que el resultado sólo termina siendo mínimamente representativo cuando se trata de "enseñar" a niños (*edutainment*), cuando se practican cosas simples o se tratan de enseñar principios, o cuando se apoya el uso con un trabajo adicional de soporte por parte de un profesor.

Por lo demás, los videojuegos de hoy son insuficientes para enseñanza superior por la carencia de los ingredientes de tutoría inteligente. La razón para esta deficiencia está en el costoso trabajo de autoría que, dentro de un videojuego, se duplica: es necesario crear la parte relativa al área que se enseña, pero también la parte lúdica.

En el próximo capítulo...

En el capítulo anterior se describieron los sistemas de enseñanza, para encontrar el problema de la *autoría*, ya fuera de *contenido* en el caso de los sistemas basados en marcos o de *conocimiento* en los sistemas de tutoría inteligente.

En este capítulo, por su parte, hemos descrito el estado actual de los videojuegos educativos, para llegar a la conclusión de que hoy en día no son

apropiados para enseñar cosas complejas debido a su escasez de ingredientes de tutoría. Normalmente la causa principal vuelve a ser la dificultad de la creación del contenido.

En el próximo capítulo, pondremos remedio a ambos problemas describiendo un modelo de tutoría integrado en juegos que pretende solventar, además, este problema de autoría.

Capítulo 4

Modelo de tutoría basado en casos para videojuegos educativos

Hasta aquí he establecido la definición de aquellas palabras que son menos conocidas, y he explicado el sentido en que quisiera se entendiesen en el siguiente discurso.

Philosophiae naturalis, Sir Isaac Newton

Resumen: En este capítulo se propone un modo de entrelazar los sistemas educativos y los videojuegos que permite una unión entre ambas componentes muy natural. Bajo el marco de trabajo de esta fusión, se describe una alternativa para la creación de contenido educativo que resulta muy beneficiosa al permitir dominios más complejos que los enseñados en los videojuegos del capítulo anterior. Además, permite mantener la enseñanza individualizada sin sobrecargar en exceso las labores de autoría.

4.1. Introducción

El capítulo 2 hizo un recorrido sobre las diferentes modalidades de apoyo a la enseñanza mediante ordenador que se han realizado a lo largo de la historia de la Informática. Aunque el gran sueño de la educación personalizada efectiva continúa estando en la mente de todos los que se dedican a este tipo de sistemas, en la práctica se siguen sufriendo muchos de los problemas de

la Inteligencia Artificial. En concreto, se tiene o bien la necesidad de gran cantidad de contenido *cableado* para crear la ilusión de inteligencia, o bien la dificultad de especificar de manera declarativa el conocimiento de un experto para poder usarlo en *razonamiento automático*.

Por su parte, en el capítulo anterior nos introdujimos en el mundo de los videojuegos. La idea de aprovechar su capacidad de motivación para usarlos como contenedores de sistemas educativos que consigan que los alumnos dediquen más tiempo a su uso resulta muy llamativa, y fue propuesta muy tempranamente. El principal problema es que resulta muy complicado conseguir una *integración intrínseca* que mantenga en equilibrio la parte lúdica y la educativa para que tanto alumnos como profesores y padres queden contentos con el resultado. Además, las escasas ocasiones en las que esto se consigue suelen estar restringidas al campo del *edutainment*, pues los niños son fácilmente engañables. Cuando nos acercamos a dominios más complejos, una buena integración es mucho más difícil.

En cualquier caso, el problema de la *integración intrínseca* atañe al *diseño*, que persigue una buena fusión de las partes pedagógica y lúdica. Otro problema es añadir *componentes de tutoría* en los juegos. De hecho, los videojuegos educativos existentes en el mercado *no proporcionan prácticamente guía* al aprendizaje. Sólo los más "sofisticados" disponen de un *orden* en la secuenciación de los contenidos, que se encuentra *cableado* al estilo de los sistemas basados en marcos. Esto deja fuera las capacidades de tutoría sofisticadas disponibles en los tutores inteligentes. Las dificultades de autoría del contenido suelen ser las culpables, de nuevo, de esta situación.

En este capítulo proponemos una alternativa para fusionar videojuegos y sistemas educativos, solventando, parcialmente, la dificultad de la autoría. Para eso es necesario encontrar un punto de unión entre ambos mundos que nos permita enlazarlos de una manera cómoda y que abra la puerta a la reutilización. Las primeras secciones del capítulo plantean un entorno general que cumple este objetivo. Una vez encontrado el punto de unión, alrededor de él se identifican varios lugares de variabilidad, que crean un *espacio de alternativas* de implementación con las que instanciar el modelo. Cada una de las opciones identificadas es descrita en profundidad en el resto de secciones. Los dos capítulos siguientes, por último, detallan sendos videojuegos educativos que encajan dentro del marco general descrito aquí, particularizando cada una de las decisiones posibles que se plantean.

4.2. Enseñanza basada en ejercicios

Las teorías más modernas sobre el aprendizaje abogan por el *aprendizaje activo*, en el que el estudiante se involucra en su propio aprendizaje y colabora con él, en lugar de mantenerse como un receptor pasivo de información. Del mismo modo, el profesor cambia de papel, pasando de ser un *contenedor de*

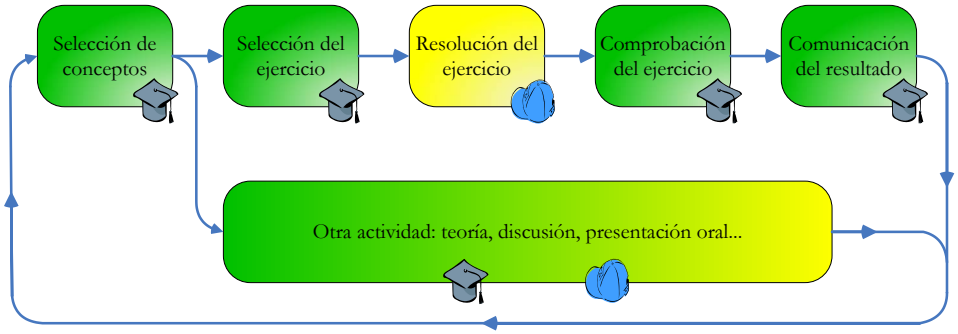


Figura 4.1: Ciclo del aprendizaje mediante ejercicios

información que debe ser transmitida a sus alumnos para convertirse en *guía* de ese aprendizaje.

El aprendizaje activo se basa en la idea del *aprender haciendo*: los alumnos construyen su propio conocimiento a través de diferentes actividades (individuales o en grupo) que, al requerir una mayor participación, potencian el recuerdo.

Las actividades beneficiosas habituales esgrimidas bajo el paraguas del aprendizaje activo son variadas. Discusiones en clase, en pequeños grupos, preguntas abiertas o debates son sólo algunas de ellas.

La que más nos interesa en nuestro contexto es, no obstante, *la resolución de ejercicios* para seguir la aproximación del *aprendizaje basado en problemas*. En ella, el alumno es enfrentado a ejercicios para los que aún no tiene todo el conocimiento necesario, con la intención de que sea él mismo el que se acostumbre a buscar los recursos que necesita para enfrentarse a situaciones nuevas, y a superarlas. Este tipo de ejercicios puede ser entrelazado con repeticiones o variaciones de casos anteriores con la intención de repasar y afianzar los conocimientos que ya tiene.

Como se ha dicho, todas estas ideas siguen requiriendo la presencia de un profesor o guía, que escoja las mejores *actividades* en cada momento, y que canalice los esfuerzos durante la resolución de cada una de ellas. Nosotros haremos hincapié en que estas actividades sean *ejercicios*, y la labor del profesor pasará por decidir a qué tipo de material (nuevo o no) enfrentar al alumno en cada momento. De esta forma, el desarrollo del aprendizaje basado en estas ideas se despliega en el ciclo mostrado en la figura 4.1. Muestra cinco etapas (Gómez Martín et al., 2005a,b):

- *Selección de los conceptos a practicar*: es realizado por *el profesor*, y consiste en elegir qué conceptos del dominio que está enseñando cree que deben ser practicados en la iteración actual. Dependiendo de la situación, podría decidir repasar algo ya conocido, repetir un tipo de

ejercicios que no se han resuelto adecuadamente, o plantear nuevos retos que requieran ideas aún no aprendidas.

- *Selección del ejercicio*: una vez tomada esa decisión, de nuevo *el profesor* escoge (o crea) el ejercicio que más se adapte a los objetivos a cubrir elegidos en la etapa anterior. Podría ser necesario modificar ligeramente el enunciado para dar variedad si la disponibilidad de ejercicios es pequeña y se quiere evitar la repetición.

- *Resolución del ejercicio*: en esta ocasión es, naturalmente, *el alumno* quien se enfrenta al ejercicio para resolverlo (o al menos intentarlo).

- *Comprobación de la solución*: el resultado es entregado *al profesor*, quien lo analiza y determina su corrección o los errores cometidos.

- *Comunicación del resultado o feedback*: el profesor comunica al alumno el resultado y le proporciona consejos sobre si debería poner énfasis en adquirir conocimientos teóricos o se le proporcionan, sencillamente, explicaciones sobre sus errores para que aprenda. Esta etapa es importante especialmente en dominios de una cierta complejidad donde el aprendizaje por prueba y error sería en otro caso demasiado lento. Al fin y al cabo, el aprendizaje por descubrimiento no puede aspirar a que el conocimiento sencillamente "florezca" en las mentes de los alumnos por propia iniciativa.

La figura muestra también una alternativa diferente a los ejercicios en cada iteración, dejando abierta la posibilidad de que la actividad realizada sea una clase teórica, una discusión en grupo, una exposición oral por parte de los alumnos, etcétera. No obstante, no nos preocuparemos en exceso de esa posibilidad aquí.

Este ciclo es muy habitual en la educación tradicional. El profesor inicia una lección con una parte teórica junto a, quizá, varios ejemplos, y luego propone una serie de ejercicios que los alumnos deben ser capaces de resolver a partir de la parte teórica explicada previamente.

La aproximación del *aprendizaje basado en problemas* pretende hacer aún más activa la tarea de aprender, eliminando, en la medida de lo posible, las clases teóricas. Una adecuada elección de los ejercicios puede conseguir que los alumnos sean conscientes de sus lagunas, y traten de resolverlas. En cierto modo, esta técnica pedagógica trata de solucionar el que Paenza (2005) considera el mayor problema de la educación en sus primeros niveles. En concreto, los docentes suelen dedicarse a dar respuestas a preguntas que los niños *no se han hecho*. Los alumnos tienen que tolerar explicaciones que no les descubren algo que les interesara en un principio, lo que resulta definitivamente muy aburrido.

Con el aprendizaje basado en problemas el profesor *plantea las preguntas*, y los alumnos deben responderlas, invirtiendo así el proceso. Como ya se ha comentado, el profesor deja de ser fuente de conocimiento para pasar a ser *guía* que informa sobre dónde buscarlo. La carencia de esta guía convertiría al proceso en una labor excesivamente ardua y, muy probablemente, frustrante y propensa a abandonos. Aunque la aproximación pura de enseñanza basada en ejercicios en los que *no* hay clases de teoría explícitas puede ser beneficiosa en muchos casos, Cooper y Sweller (1987) demostraron que no lo es tanto durante las primeras etapas del aprendizaje de un dominio. La explicación a este fenómeno es la *sobrecarga* sufrida por la completa carencia de conocimientos, que hace que los alumnos no dispongan de *heurísticas* sobre cómo abordar los primeros problemas.

Independientemente de si se utiliza un aprendizaje basado en problemas "puro" o no (que, en cualquier caso, es poco corriente), en el contexto de la escolarización tradicional normalmente se sobreentiende que las clases serán compartidas por un grupo variado de alumnos. Esto daña las etapas llevadas a cabo por el profesor. En primer lugar, la elección de los ejercicios se realiza en función del estado global de todos los alumnos, de modo que el mismo ejercicio estará mejor o peor adaptado dependiendo de lo cerca que cada alumno particular esté de la "media" de la clase. Del mismo modo, las últimas etapas en las que el profesor analiza el rendimiento del alumno y le proporciona ayuda específica para su caso se complican por la estrechez del tiempo. No es de extrañar, por lo tanto, las pegas que, respecto al aprendizaje de contenidos, planteaba Bloom (1984).

Debido a esto, y recordando las promesas del capítulo 2 sobre la capacidad de la educación asistida por ordenador de proporcionar educación individualizada, podemos plantearnos cómo encajan cada uno de los tres tipos de sistemas principales (simuladores, sistemas basados en marcos y tutores inteligentes) dentro de este ciclo.

Un simulador abarca únicamente la etapa de resolución del ejercicio por parte del alumno. Obviamente, en este caso el ejercicio puede llegar a ser una situación muy compleja y su resolución requerir mucho tiempo, diferentes pasos y tener un final increíblemente abierto. Pero dado que un simulador *no* dispone de las dos primeras y últimas etapas, sigue siendo necesario la existencia de un tutor que guíe el aprendizaje, que plantee ejercicios y que analice el desempeño del alumno para proporcionarle críticas constructivas que le hagan mejorar. Esta última etapa a veces se conoce con el término militar anglosajón *debriefing*.

En el caso de los sistemas basados en marcos, la mayor parte de la carga de contenido se refiere a *parte teórica*. Los escasos ejercicios (que suelen ser tipo test o de algún otro tipo de entrada sencilla) tienen como finalidad averiguar si el alumno ha entendido los conceptos clave, y en ese caso pasar al siguiente objeto de aprendizaje.

A pesar de eso, estos sistemas *encajan* en el esquema anterior. La mayor parte de las iteraciones irán por la rama de las explicaciones teóricas, pero desde luego el sistema es capaz de decidir *automáticamente* qué hacer en cada momento en función del perfil del estudiante. Por ejemplo, en la primera iteración mostrará teoría, y en la inmediatamente posterior realizará un pequeño test. Dependiendo del caso, se puede (o no) mostrar información sobre los errores con explicaciones asociadas a cada respuesta inválida. En el comienzo de la tercera iteración, se decidirá si volver a plantear el mismo cuestionario al alumno, repetir la parte teórica, o avanzar.

Debido a la simplicidad de estos sistemas, las decisiones llevadas a cabo por la primera etapa están *cableadas* en el propio curso. Los creadores del contenido habrán previsto todas las posibles respuestas del alumno a cada uno de los test, y habrán codificado de algún modo la reacción que el sistema habrá de tener al encontrarse cada una de ellas.

Debido a la simplicidad de los ejercicios, los sistemas basados en marcos raramente potencian el aprendizaje activo y, con tanta carga teórica, menos aún el aprendizaje basado en problemas puro. Aun así, son adecuados para muchos dominios en los que tiene mucho más peso la parte teórica que la práctica. En este tipo de contextos, los sistemas basados en marcos permiten la realización *completa* de las iteraciones de manera individual y sin necesidad de un profesor humano que controle el aprendizaje, tal y como ocurría en los simuladores.

En realidad, el principal problema de los sistemas anteriores es que al estar todo el conocimiento codificado de manera *procedimental*, la parte activa del aprendizaje no puede ser excesivamente amplia. De otro modo, el abanico de posibilidades que habría que contemplar sufriría un incremento exponencial, y el modelo dejaría de escalar bien. Si se desea enseñar dominios donde la práctica resulta más importante que la teoría usando este tipo de sistemas, es necesario recurrir, de nuevo, a un profesor humano. En este caso, la enseñanza puede seguir apoyada por el ordenador para la parte teórica, pero el profesor se encargará de, al menos, las dos últimas etapas (comprobación de la solución y comunicación de los resultados) de los ejercicios clave más activos. En función del resultado, el profesor proporcionará información específica para mejorar, o "desbloqueará" la siguiente parte del curso para que el sistema permita continuar con los siguientes episodios de aprendizaje.

Incluso con el apoyo de profesores en los momentos clave, los sistemas basados en marcos se demuestran incapaces de proporcionar una atención constante del alumno *durante* la resolución del ejercicio. Si éstos resultan ser más complejos que meros cuestionarios (acercándonos al tipo de resolución mostrado en un simulador), resulta inviable tener previstas todas las opciones posibles. En este caso, tendremos que hacer uso de los tutores inteligentes de la sección 2.9.

En ellos, el conocimiento procedimental sobre el dominio que se ha ca-

bleado *codificando decisiones* en los sistemas basados en marcos desaparece, y se proporciona *conocimiento declarativo* que es utilizado por módulos de inferencia que aplican la información de acuerdo a la situación actual concreta. Esto permite *reaprovechar el conocimiento entre ejercicios*, y ser capaz de interpretar y responder a situaciones concretas no previstas individualmente por los diseñadores, sino mediante generalizaciones.

Gracias a esto, resulta posible sofisticar el tipo de ejercicios, que pueden resultar mucho más complicados y, aun así, el sistema será capaz de interpretar el resultado. Además, se hace viable *fusionar* las tres últimas etapas. Si el sistema proporciona un *entorno* en el que resolver los ejercicios (en lugar de quedar a la espera de la respuesta final), el sistema puede monitorizar y proporcionar información contextualizada *durante* la resolución del ejercicio (y no a posteriori). En función de cómo y cuando se lleve a cabo esta entrega de información, podríamos incluso conseguir que el sistema *guiara* en lugar de que *enseñara*, tal y como persigue el aprendizaje basado en problemas.

Antes de analizar cómo realiza cada una de las etapas del ciclo un tutor inteligente, recordemos rápidamente los cuatro tipos de conocimiento vistos con detalle en la mencionada sección 2.9:

- *Conocimiento del dominio*: contiene conocimiento declarativo sobre el dominio que se enseña. En el caso de la enseñanza basada en ejercicios, el *módulo experto* que lo utiliza debe ser capaz de generar ejercicios y resolverlos. La forma en la que esto se realiza debería ser lo más "transparente" posible para el resto de los módulos, de modo que puedan obtener información sobre las *causas* por las que algo se ha resuelto de un determinado modo, o por qué una solución del usuario no es correcta.

- *Conocimiento del usuario*: lleva la pista sobre lo que el estudiante sabe y sobre lo que no. El *módulo* que lo utiliza (módulo del usuario) es el encargado de actualizar ese modelo. Eso le obliga a deducir las *causas* de las equivocaciones, por lo que se pedía la "transparencia" del módulo experto anterior. Esto entra en el campo de la *generación de teorías* pues a partir de unos hechos (pasos del estudiante) es necesario generar una teoría sobre la causa (el modelo del estudiante con lo que sabe y lo que no).

  En realidad se puede hablar de modelo de usuario a *largo* y a *corto plazo*. El primero mantiene la información general sobre lo que el estudiante ya sabe y lo que aún no, y se usa para tomar decisiones sobre lo siguiente a practicar. El modelo a corto plazo guarda información sobre el desarrollo del ejercicio actual y se usa para la ayuda contextual, buscando causas a los errores para tomar medidas.

- *Conocimiento pedagógico*: debe tener información sobre las teorías del aprendizaje para aplicarlas al buscar la mejor forma de enseñar. Por

	Selección de conceptos	Selección del ejercicio	Resolución	Comprobación de la solución	Comunicación del resultado
Dominio	Currículo	Generación		Resolución	
Usuario	Largo plazo			Corto plazo	
Pedagógico	Nivel global				Nivel local
Interfaz					Generación

Tabla 4.1: Uso del conocimiento del ITS en la enseñanza mediante ejercicios

desgracia, es el gran olvidado. Teóricamente debería ser independiente del dominio, pero en la práctica, cuando aparece, nunca lo es. Con ese conocimiento, y el modelo del estudiante, el *módulo* pedagógico se encarga de dos cosas:

- *A nivel global*: decidir qué hacer en el siguiente episodio de aprendizaje: ¿proponer un nuevo concepto? ¿Repasar alguno previo? ¿Introducir teoría nueva? ¿Mediante qué tipo de presentación? Esta decisión se toma en función de las supuestas *teorías pedagógicas* (codificadas en el conocimiento pedagógico); inicialmente no se preocupa sobre qué cosa nueva introducir, sino si hacerlo o no para evitar sobrestimar o subestimar al alumno por ejemplo.
- *A nivel local*: decidir cuando interrumpir. El módulo del usuario habrá decidido que algo va mal, y ahora es el momento de decidir si interrumpimos o no al estudiante y en qué grado (un mero aviso, un consejo o una explicación teórica), también en función de las teorías del aprendizaje.

En ambos casos se debería conocer el estilo de aprendizaje del estudiante.

- *Conocimiento del interfaz*: permite la generación superficial de las explicaciones. En la práctica, tenía mucho sentido cuando la comunicación se producía a través del procesamiento de lenguaje natural. Hoy en día se encuentra integrado en los agentes pedagógicos.

Si cruzamos los cuatro tipos de conocimiento con las cuatro etapas realizadas por el ordenador del ciclo propuesto, encontramos los puntos de unión mostrados en la tabla 4.1. Cada columna representa una etapa del ciclo, y cada fila un tipo de conocimiento. La columna asociada a la etapa de resolución del ejercicio aparece completamente sombreada, dado que el sistema no interviene en ella[1]. También aparecen sombreadas las celdas que cruzan

[1] La resolución es realizada por el alumno, aunque durante ésta, el sistema pueda estar monitorizándole para detectar errores y ayudarle (dos últimas etapas).

una etapa con un tipo de conocimiento que no es utilizado en ella.

4.3. Los videojuegos y la enseñanza mediante ejercicios

Una vez analizado el uso de los diferentes tipos de conocimiento utilizado en las etapas de los sistemas de enseñanza basada en ejercicios, retomamos de nuevo la idea del uso de videojuegos en educación que describimos en el capítulo 3. No tenemos noticia de ninguno de estos videojuegos que disponga de manera explícita de alguno de los tipos de conocimiento descrito anteriormente. Muchos abogan por una enseñanza activa basada en práctica (es decir, ejercicios), y sin embargo no se mantiene perfil del estudiante, pocos *evalúan*, y muchos menos *explican*. En esta sección planteamos nuestro modelo de fusión de sistemas educativos y videojuegos que crea un marco de trabajo en el que incluir todas las características de los tutores inteligentes que hoy se echan de menos en los videojuegos. Para eso, comenzaremos revisando el funcionamiento general, percibido por el usuario, de los programas de entretenimiento.

Aunque algunos juegos no disponen de un final establecido (por ejemplo el conocido Tetris, y muchos simuladores), la mayoría poseen uno o varios modos de ser terminados. El contenido se termina, y el juego se considera superado.

Esto es lo que ocurre, al menos, en los videojuegos basados en niveles. En ellos, el jugador se enfrenta a niveles progresivamente más complicados, quizá con enemigos especialmente significativos en dificultad al final de cada nivel (los conocidos con la palabra inglesa *bosses*).

La figura 4.2 muestra el esquema del funcionamiento de un videojuego de este tipo. Las etapas de las que está compuesto son:

- *Creación del perfil*: se pide el nombre del jugador y se inicia el juego. Esta posibilidad depende del videojuego, pero la idea es poder tener varias partidas en curso en la misma máquina con perfiles diferentes. Inmediatamente después de la creación del perfil puede proponerse un breve tutorial, o una fase sencilla, en la que mostrar la parte más básica de manejo del juego. Esto permite el *instalar y listo* ahorrando la lectura de las grandes olvidadas: las instrucciones.

- *Elección del siguiente nivel*: en función del punto de la partida, el sistema plantea un nivel u otro. Naturalmente, aparte de tras el tutorial, es posible llegar aquí después de haber jugado un nivel anterior, o tras cargar una partida grabada. Dependiendo del juego, el usuario puede o no tener la posibilidad de intervenir en esta etapa. Por ejemplo, algunos videojuegos permiten repetir niveles ya jugados en busca de secretos no

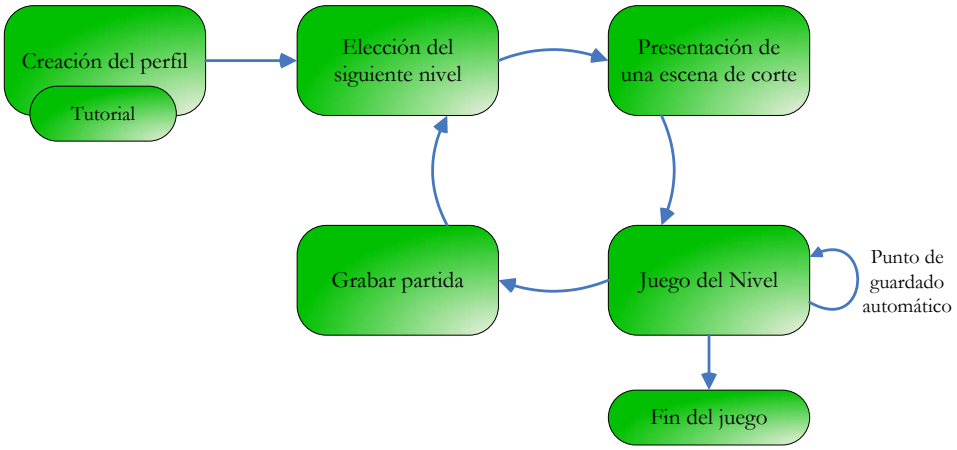


Figura 4.2: Ciclo de un videojuego

descubiertos. En otro caso, esta etapa ocurrirá internamente al sistema sin que el jugador sea ni siquiera consciente de ella.

- *Presentación de una escena de corte*: dependiendo del nivel y la situación de la historia, podría mostrarse una escena de corte (*cut scene*) que describa un poco el nivel contando qué se espera del jugador, o las causas por las que se ha llegado a él.

- *Juego del nivel*: el usuario se pone, finalmente, a jugar el nivel concreto dentro del correspondiente entorno. A menudo, el sistema realiza grabaciones no permanentes del progreso, de modo que si el jugador sufre algún percance, pueda volver a una situación intermedia anterior en lugar de tener que repetir el nivel completo.

- *Grabar partida*: tras superar el nivel por completo, normalmente la partida se grabará para poder volver a ese estado en otro momento. Los puntos de grabado automáticos *internos* al nivel suelen ser *volátiles* y no sobreviven al cierre del programa, por lo que esta etapa resulta importante en los juegos mínimamente largos.

- *Fin del juego*: si el nivel es superado y era el último, el juego se considera superado, se le indica al usuario y se termina.

Este modo de proceder es increíblemente parecido al del aprendizaje basado en ejercicios cuando equiparamos *un ejercicio con un nivel*. En realidad, la separación en las etapas de este tipo de enseñanza descrita en la sección anterior se realizó pensando en maximizar su aplicabilidad, tanto a la enseñanza tradicional con un profesor humano como a los sistemas de apoyo a la enseñanza del capítulo 2. Esto, además, nos permitió cruzar las etapas

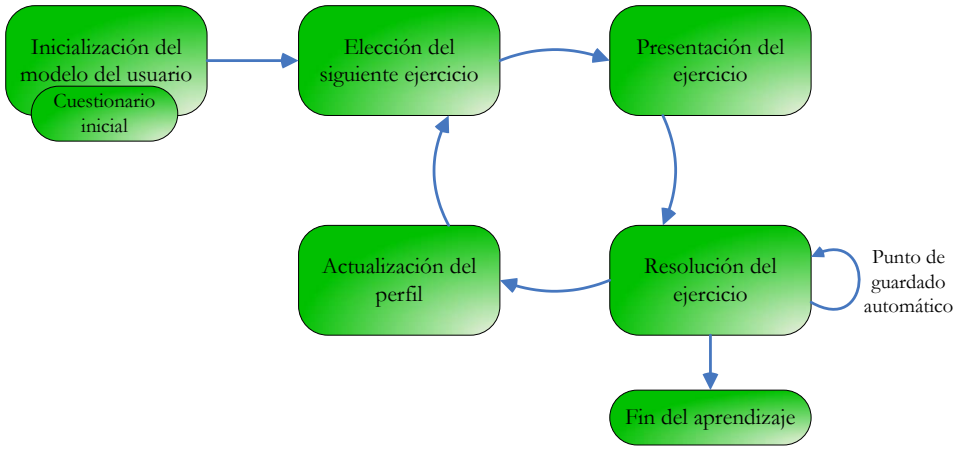


Figura 4.3: Aprendizaje mediante ejercicios (punto de vista del alumno)

de manera coherente con los cuatro tipos de conocimiento de la arquitectura clásica de los tutores inteligentes. Pero si pensamos en el funcionamiento general tal y como lo ve un usuario cuando los ejercicios son resueltos en un entorno proporcionado por el propio sistema, las similitudes con los videojuegos basados en niveles saltan a la vista (figura 4.3):

- *Inicialización del modelo del usuario*: el estudiante se da de alta en el sistema y se recoge su información. Como primera medida, se le podría realizar un pequeño cuestionario para averiguar sus conocimientos y actualizar el modelo del usuario con la información extraída.

- *Elección del siguiente ejercicio*: el sistema escoge el siguiente ejercicio. Anteriormente dividíamos esta etapa en dos fases (selección de los conceptos y selección o creación del ejercicio adecuado a ellos) para poder ajustar con mayor granularidad el uso de los diferentes tipos de conocimiento.

- *Presentación del ejercicio*: dependiendo del tipo de ejercicio, el sistema podría mostrar información previa necesaria para su correcta resolución.

- *Resolución del ejercicio*: dentro del entorno de aprendizaje, el usuario resuelve paulatinamente el ejercicio. El sistema puede monitorizar las acciones, y tomar medidas cuando detecta que se equivoca proporcionando ayuda contextualizada. Si la situación de resolución degenera hasta más allá de un límite, podría recuperarse un estado intermedio anterior para que el alumno reemprenda la resolución desde un lugar correcto conocido.

Juegos	**Aprendizaje basado en ejercicios**
Inicio del juego	Registro de un nuevo usuario
Tutorial del juego	Cuestionario de conocimientos previos
Juego de un nivel	Resolución de un ejercicio
Grabación de la partida	Actualización del modelo de usuario
Carga de una partida	Recuperación del modelo del usuario
Finalizar el juego completo	Aprendizaje completo conseguido
Imposibilidad de terminar un nivel	Resolución incorrecta de un ejercicio
Vuelta a un punto de guardado automático	Vuelta a un estado seguro tras muchos errores

Tabla 4.2: Relación entre un juego y el aprendizaje mediante ejercicios

- *Actualización del perfil*: en función del modo en el que el ejercicio haya sido resuelto, el modelo del estudiante a largo plazo es actualizado para tomar nota de las cosas que ha aprendido o que insiste en hacer mal.

- *Fin del aprendizaje*: si se determina que el alumno ha aprendido ya todo lo que el sistema es capaz de enseñarle, se le felicita por ello y se termina.

Con este planteamiento, la relación es casi uno a uno, tal y como se muestra en la tabla 4.2. Esto descubre que los videojuegos realizan una elección *secuencial* del siguiente nivel, cuyo orden ha sido codificado de antemano en el propio videojuego. Además, esa elección es atómica, no como ocurría en nuestro ciclo original de los sistemas educativos en el que aparecía separada en dos etapas.

En realidad, todo esto nos da un marco sobre el que buscar otras relaciones. Éstas se podrán utilizar para buscar modos de hacer a los tutores inteligentes más entretenidos (Gómez Martín et al., 2004b), o a los videojuegos un poco más sencillos si se incorporan ideas de los tutores inteligentes (Gómez Martín et al., 2004a).

Por ejemplo, en los videojuegos llegar al final es seguramente una de las motivaciones principales para muchos usuarios. No obstante, ese gran "premio" es, según Rollings y Morris (2003), un objetivo muy lejano en el momento de iniciar la partida. Para conseguir mantener la atención del jugador, algunos juegos añaden pequeños premios al final de cada nivel o grupo de niveles. Quizá el ejemplo más conocido sean los personajes bailarines en la implementación de arcade original del Tetris. También dentro de esta categoría entran las escenas de corte ya mencionadas que, aparte de hacer evolucionar la historia, pueden hacer uso de técnicas multimedia que hagan que el jugador quiera ver más. Naturalmente, estos premios "visuales" no son suficientes, y el juego debe ser entretenido en sí mismo, aunque la experiencia muestra que sí suelen potenciar, aunque sea de manera discreta, que los usuarios jueguen más.

Otra alternativa para mantener el interés del jugador es desbloquear características no activadas originalmente. Un ejemplo claro son los minijuegos que, de hecho, a veces se utilizan como reclamo publicitario tal y como atestigua la carátula trasera de *Rayman 3*. La última versión para Wii (*Rayman Raving Rabbids*) continúa en esa linea y según avanza el juego se desbloquean, por ejemplo, elementos para el traje del protagonista.

Dada la relación entre el funcionamiento de videojuegos y sistemas educativos basados en ejercicios, esta idea podría importarse desde los primeros hasta los segundos. Así, *a nivel de diseño* en la integración de la componente lúdica y educativa, se podría buscar la manera de premiar con algún tipo de elemento adicional al jugador según va resolviendo ejercicios. En Gómez Martín et al. (2004b) detallamos otros muchos puntos de mejora de este tipo.

Una vez enunciados los puntos de unión entre los dos tipos de sistemas que nos proponemos unir (videojuegos y sistemas educativos orientados a ejercicios), recuperaremos en las siguientes secciones las etapas tal y como se describieron en la sección 4.2. En concreto, describiremos cada una de las cinco etapas un poco más en detalle, viendo alternativas para su desarrollo y la posible relación que tiene con los videojuegos. Como dijimos en la introducción del capítulo, la disponibilidad de estas alternativas, crea un espacio de posibilidades a la hora de desarrollar videojuegos educativos que sigan el modelo de fusión basado en el emparejamiento ejercicio–nivel. Los dos capítulos siguientes plantearán dos de tales videojuegos educativos que instancian dicho modelo de diferente forma tomando distintas decisiones en cada punto de variabilidad descrito aquí.

4.4. Selección de los conceptos a practicar

En un sistema de enseñanza basado en ejercicios, el primer paso es decidir qué objetivos pedagógicos se quieren cumplir en la siguiente iteración. En la educación tradicional estos objetivos son elegidos por el profesor. Para eso, debe ser consciente tanto de los conocimientos del (o los) alumnos, como de la *programación de la asignatura* planificada a principio de curso en función de los planes de estudio y de las preferencias en su modo de dar clase.

En la enseñanza basada en ordenador, la aproximación más sencilla de implementación es no preocuparse por esta etapa, y asignar la responsabilidad de la elección al propio usuario. Como ya dijimos, esta situación es la que se produce en los simuladores.

Si queremos añadir la posibilidad de que la aplicación escoja el objetivo del siguiente episodio de aprendizaje necesitamos desgranar el dominio que se enseña en diferentes partes para tener elementos individuales entre los que elegir. Los sistemas basados en marcos consiguen esto troceando el contenido en *objetos de aprendizaje* y la elección del objetivo pedagógico se hace de manera inseparable a la selección del material (que nosotros hemos llevado

a la siguiente etapa).

En este tipo de sistemas, el modo en el que se determina el siguiente paso a realizar es, por tanto, mediante el recorrido *del grafo* de esos objetos de aprendizaje realizado por el diseñador del material. Es por eso por lo que comentábamos que en estos sistemas se *codifican las decisiones*, pues la elección es realizada por *el experto que hace el contenido* y se cablea dentro de ese grafo.

Es en los tutores inteligentes donde la separación entre las dos etapas cobra especial sentido. En este caso, el dominio suele ser diseccionado en diferentes *conceptos*, pero *sin* asociar (todavía) un concepto con el material que lo enseña o practica. Además, es necesario disponer de información sobre relaciones entre ellos: qué conceptos dependen de otros, cuales son más fáciles, cuales son similares o van siempre juntos, etcétera. Toda esta información se suele considerar parte integral del *conocimiento del dominio*, aunque a veces es vista como un componente independiente dentro del tutor inteligente. Es lo que hace, por ejemplo, Kopec y Thompson (1992, página 93) considerándolo un tipo de conocimiento diferente al del dominio y llamándolo *conocimiento del currículo*.

La forma en la que se guarde toda esta información afectará al *modelo del estudiante*. Lo más habitual es que éste se guarde mediante *plantillas*, de modo que el sistema siga la pista de qué *conceptos* sabe ya el alumno, y cuales no. Esta idea puede mejorarse si en lugar de marcar de manera discreta (sí o no) si un concepto es conocido, se almacenan estimaciones porcentuales y, además, se mantiene una separación entre conocer un concepto, no conocerlo y *tener una idea equivocada* sobre él (que es aún peor que no saberlo).

Además, este *conocimiento del currículo* hará explícito el mejor orden para enseñar una determinada materia, extraído a partir de las *precondiciones* de cada uno de los conceptos. Viéndolo así, podría pensarse que también en él *codificamos decisiones* en cuanto a la elección del siguiente episodio de aprendizaje, pues esas precondiciones también crean un *grafo de conceptos* que habría que seguir.

La diferencia principal es que ahora los conceptos *no* están integrados con el contenido directamente en ese grafo, lo que permite disponer de mecanismos de inferencia para recorrerlo según convenga. De hecho, la decisión sobre el objetivo pedagógico es independiente del material que se terminará usando, que podría decidirse a posteriori en función de, por ejemplo, las preferencias del estudiante. Así, el conocimiento del currículo se utiliza junto con el modelo del estudiante (a largo plazo) mencionado antes para escoger la mejor opción en cada momento. En cierto modo, el grafo de conceptos se "cruza" con la plantilla del modelo del usuario para decidir los conceptos que se deberían poner en práctica en la siguiente iteración.

En ese cruce se hace uso del *conocimiento pedagógico*, que, en teoría, tiene información sobre la mejor forma de enseñar. Por ejemplo, podría indicar al

sistema que es mejor no pasar a un concepto superior si el que le sirve como base no se conoce más allá de un umbral de, pongamos, un 75 %. Como ya hemos comentado, idealmente esta información debería ser independiente del dominio para así poderla reaprovechar entre tutores de diferentes áreas pero, en la práctica, no suele ser así.

Una alternativa para la representación del conocimiento pedagógico es hacer uso de CBR (*Case-based reasoning*, Razonamiento basado en casos), en el que en lugar de guardar reglas se mantienen *experiencias* anteriores (Riesbeck y Schank, 1989; Aamodt y Plaza, 1994). En el caso del conocimiento pedagógico, se almacenaría información sobre decisiones anteriores que nos relacionen una situación concreta con la decisión que se tomó y el resultado (bueno o malo) que se obtuvo. Reyes y Sison (2001, 2002) presentan un sistema de este tipo, aunque queda envuelto dentro de un sistema multiagente que lo hace bastante difícil de seguir.

En esta etapa, también se decide el tipo de actividad que se realizará a continuación. Es decir, una vez escogido el o los conceptos a practicar, es necesario planificar el resto de la iteración. Un profesor humano tomaría esta decisión en función de muchas cosas, como preferencias personales, particularidades de los conceptos escogidos o nivel de participación general del grupo de alumnos. En un sistema digital, de nuevo, habría que utilizar el conocimiento del currículo, el pedagógico y el modelo del estudiante. La información sobre los conceptos escogidos deberían dar información sobre los mejores modos de enseñar o poner en práctica ese área del dominio enseñado. El modelo del estudiante particularizaría la situación a un alumno concreto, y ambas partes serían analizadas por el conocimiento pedagógico para escoger la mejor actividad pensando en ese usuario.

En la práctica, nosotros no nos preocupamos de esta decisión porque *siempre usamos ejercicios*, por lo que no hay posibilidad de escoger otra actividad diferente. En cualquier caso, en los sistemas educativos digitales el número de posibilidades tampoco es excesivamente alto si queremos *que el sistema pueda evaluar*. Debido a esto, las opciones se restringen prácticamente a lecciones teóricas y a ejercicios, quedando fuera otras tareas clásicas del aprendizaje activo como la exposición oral o las discusiones en grupo.

Cuando comparamos los sistemas de enseñanza con los videojuegos en la sección anterior, dijimos que en estos últimos también se realiza una selección de la siguiente iteración a realizar, pero esa decisión está cableada en la secuencia de niveles ordenada por los diseñadores.

Tal y como nos cuenta Gee (2004), para poder superar cualquier juego es necesario un *aprendizaje*, y éste deberá realizarse, como cualquier otro, de manera gradual. La ordenación de la secuencia de niveles de un juego está decidida de antemano (*cableada*) pero con el suficiente cuidado como para que ese aprendizaje *sea paulatino*. Esta ordenación prefijada de antemano puede compararse con lo que ocurría en los sistemas basados en marcos,

puesto que, además, los videojuegos tampoco separan los "conceptos" del "contenido" (niveles). Los videojuegos comerciales podrían intentar *adaptarse* a los jugadores esporádicos si fueran conscientes de este hecho y adaptaran el nivel siguiente a las destrezas del jugador de modo que la *experiencia* vivida por todos ellos fuera, independientemente de sus habilidades, relativamente similar.

Al mezclar videojuego y educación, en función del tipo de integración entre ambas partes, la elección de los conceptos a practicar puede, de hecho, afectar también a la dificultad de la parte lúdica. Para equilibrarlo, aquí podríamos también tomar decisiones sobre la dificultad de los componentes adicionales no directamente relacionados con los aspectos educativos del sistema. No obstante, equilibrar la dificultad de un videojuego es un trabajo muy laborioso que suele realizarse de manera artesanal. En Gómez Martín et al. (2006b) planteamos un modo de sistematizar este proceso utilizando FCA (*Formal Concept Analysis*, Análisis Formal de Conceptos).

4.5. Elección del siguiente ejercicio

Una vez seleccionado qué conceptos queremos poner en práctica en la iteración actual gracias al conocimiento del currículo, pedagógico y modelo del usuario, es necesario encontrar un ejercicio adecuado que lo consiga. Aquí entra en juego de nuevo el conocimiento del dominio, cuya labor principal, decíamos, era precisamente ser capaz de proporcionar y resolver los ejercicios del área enseñada.

Tradicionalmente se han propuesto dos modos principales para su implementación. La primera de ellas es hacer uso de un *sistema experto* que, utilizando algún tipo de representación declarativa, haga explícito el *conocimiento* del área de modo que pueda utilizarse para *crear al vuelo* ejercicios *coherentes* que pongan en práctica los conceptos proporcionados por la etapa anterior. Es lo que hace, por ejemplo, Kumar (2002) para disponer de ejercicios ilimitados en su tutor inteligente sobre depuración del lenguaje de programación C++.

El principal problema de esta aproximación es la dificultad de la creación del conocimiento necesario para conseguir esta capacidad de invención de ejercicios. Esto es en realidad una crítica general a todos los tutores inteligentes. El paso de los sistemas basados en marcos a los ITS es muy complejo porque es difícil disponer de expertos del dominio que se enseña que *además* sean capaces de hacer explícito ese conocimiento del modo en el que un ordenador es capaz de comprenderlo y utilizarlo.

Naturalmente, este problema no surgió, ni mucho menos, por primera vez en los tutores inteligentes, y era conocido ya en el área de la Inteligencia Artificial. La alternativa defendida por muchos investigadores es el CBR ya mencionado, con el que la extracción de conocimiento es mucho más sencilla

por ser *episódica*, algo que en general a los expertos humanos les resulta mucho más fácil de hacer.

El razonamiento basado en casos encaja perfectamente en los tutores inteligentes que nos ocupan. La idea es mantener una *base de ejercicios*, en la que se mantenga qué conceptos ponen en práctica cada uno de ellos. Cuando la etapa anterior nos indica qué se quiere practicar, para conseguir el ejercicio adecuado convertimos la *generación* (usada cuando se dispone de un modelo experto) en *elección* dentro de una batería de opciones mantenidas en la base de ejercicios (Stottler, 2000; Stottler et al., 2001a).

El uso de CBR aquí a veces resulta, sorprendentemente, un poco confuso para aquellos que están habituados al propio CBR. Normalmente esta técnica se utiliza para *resolver problemas*, como la colocación de piezas en un horno (Hennessy y Hinkle, 1992) o *help desk*. En ese caso, la búsqueda sobre la base de casos se realiza a partir de una descripción del problema, y se obtiene su solución. En nuestro caso, la consulta se realiza utilizando los conceptos a practicar, y la respuesta del sistema *es un problema* (ejercicio para el alumno). Cómo se obtiene la solución de dicho problema no lo estamos considerando aún aquí; trataremos ese tema en la sección 4.7.

Con este cambio en el paradigma de la Inteligencia Artificial utilizado se trata de "esquivar" el problema de la autoría de conocimiento en lo que se refiere a *cómo* hacerlo. Dado que a menudo es muy complicado modelizar con, por ejemplo, reglas un determinado dominio, olvidamos la idea de la generación al vuelo de ejercicios, pero forzamos al experto a crearlos manualmente. En cierto modo este cambio vuelve un poco atrás. El conocimiento *explícito* esgrimido como punto fuerte de los sistemas inteligentes vuelve a convertirse en este punto en *mero contenido*: el experto se ve obligado a crear una amplia cantidad de ejercicios para conseguir que sean siempre suficientes independientemente del alumno al que haya que enseñar.

Este problema aparece también en los videojuegos. Los *niveles* de un juego pueden crearse de manera manual por parte de un diseñador (equivalente a la aproximación basada en casos), o de manera *procedimental*, generando el nivel automáticamente al vuelo (equiparable a la aproximación con un sistema experto). La generación manual es la más habitual, y de hecho los equipos de desarrollo de los videojuegos requieren muchos más diseñadores y grafistas (que generan el contenido) que programadores. En cierto modo, resulta mucho más *controlable*. La generación procedimental cambia el peso, pues la parte importante vuelve al desarrollo, que debe implementar modos para que el propio programa sea capaz de generar los niveles, en lugar de tener una masa de diseñadores haciendo ese trabajo durante la producción del juego.

El uso de una u otra alternativa depende fuertemente del tipo de juego, pero también del número de niveles que queramos tener. Con la generación manual, la relación entre este número y el coste es lineal: hacer el doble de

niveles cuesta dos veces más. Sin embargo, con la generación procedimental *cuesta mucho construir los primeros niveles*, pues se necesita introducir en un ordenador el modo en el que los diseñadores humanos hacen su trabajo. Sin embargo, una vez afinado el sistema que los genera, la creación de niveles adicionales *es muy barata*. Por lo tanto, en función de cuantos niveles queramos tener, y el coste de conseguir la generación automática (que varía con el tipo de juego y cuyo éxito, en cualquier caso, es difícil de anticipar) se tomará un camino u otro.

En nuestra opinión, la lucha entre *conocimiento* y *contenido*, sin embargo, tiene su solución en, como siempre, el punto medio: los sistemas llamados KI-CBR (*Knowledge Intensive Case-Based Reasoning*, Razonamiento basado en casos rico en conocimiento) descritos, por ejemplo, por Aamodt (1990). La idea es que el experto continúe proporcionando una batería de ejercicios pero utilizando un "lenguaje" bien definido que proporcione conocimiento declarativo específico del dominio que pueda ser usado por el sistema. Esto permite *reutilizar* parte del conocimiento *entre múltiples ejercicios*. Además, haciendo uso de la fase de *adaptación* del ciclo CBR, el experto podría limitarse a introducir *ejercicios prototípicos* y que el sistema se encargue de crear variaciones y mezclarlos entre sí. En el capítulo 6 se ejemplifican estas ideas en varios momentos.

Esta aproximación consigue lo mejor de las otras dos alternativas. La carga de autoría de ejercicios queda aliviada, a costa de una fase de creación de conocimiento declarativo similar a la necesaria en el caso de un sistema experto, *pero mucho más ligera*. Además, al disponer de conocimiento declarativo adicional más allá de los simples casos, es mucho más viable desarrollar la fase de adaptación del ciclo CBR, lo que amplia la diversidad. Esto pretende paliar el problema de una escasa batería de ejercicios (que, según Martin y Mitrovic (2000), es un gran problema) pero sin incurrir en un coste prohibitivo en autoría.

Este punto medio podría ser también extrapolable al caso de los niveles de los videojuegos tradicionales. En lugar de decantarse por niveles manuales o procedimentales, la opción mixta consistiría en repartir el trabajo entre los diseñadores haciendo niveles prototípicos, y desarrolladores que añaden una pizca de variación y mezcla entre ellos para que ninguno sea igual.

En lo que se refiere a la *unión* entre la parte educativa y la parte lúdica, de nuevo depende en gran medida del tipo de integración que se haya decidido en el diseño del videojuego educativo. Aunque la simetría entre ambos mundos que nosotros planteamos aquí aletea alrededor de la similitud entre un ejercicio y un nivel, el modo en el que esto se implemente posteriormente dependerá de cada situación. En lo que se refiere a la implementación, el propio nivel podría contener integrado el ejercicio, de modo que la generación (sistema experto) o elección (base de casos) de uno lleve implícito la del otro.

Otra opción es que el sistema escoja el ejercicio de manera independiente, y luego exista un modo (automático) de relacionar un ejercicio con un nivel en función del diseño y el tipo de integración. En la sección 4.9 revisaremos con un poco más de detenimiento estas alternativas.

4.6. Resolución del ejercicio

Mediante las dos etapas anteriores se habrá elegido el ejercicio al que enfrentar al estudiante. En la educación tradicional de lápiz y papel, habitualmente el alumno deberá resolver el ejercicio por sí mismo, averiguando la mejor manera de conseguirlo.

En el aprendizaje basado en problemas, esta aproximación es insuficiente. Dado que las exposiciones teóricas por parte del profesor quedan reducidas a su mínima expresión, el alumno únicamente cuenta con su intuición y el conocimiento que pueda descubrir para conseguir solucionar el problema. Por tanto, dejar al alumno completamente a su suerte contra el ejercicio puede alargar mucho el tiempo necesario para resolverlo e incluso crear frustración, salvo que se cuente con la suficiente motivación interna como para superar estos problemas.

El aprendizaje que se produce en cualquier videojuego comercial es precisamente de este tipo. Dado que la mayor parte de las veces se cuenta con esa motivación, es posible asumir que los usuarios tendrán las suficientes ganas como para, al menos, intentarlo. Sin embargo, en la enseñanza clásica esto rara vez es así, por lo que es necesario un profesor que, al menos, *guíe* al alumno realizando un seguimiento durante toda la resolución del problema.

Si queremos conseguir esta ayuda en un sistema educativo, es imprescindible que exista la posibilidad de que el ordenador tenga acceso a los pasos que el estudiante da durante la resolución. Es decir, no es suficiente con recibir *directamente la solución*, sino que el proceso completo es importante. El sistema debe, por tanto, proporcionar un *entorno de aprendizaje* dentro del cual el estudiante resuelve el ejercicio. De esta forma, es posible seguir los pasos para sacar conclusiones.

En este sentido, aparecen dos modelos de funcionamiento. El primero (y quizá más sencillo) es esperar a que el estudiante resuelva *completamente* el problema, y luego analizar el registro de pasos y el resultado final como un todo, para sacar conclusiones. Este funcionamiento es especialmente interesante para aquellos casos en los que tenemos disponible el entorno de aprendizaje (por ejemplo, un simulador construido anteriormente), y queremos utilizarlo para enseñanza automática. Lo único que necesitaremos será añadir la capacidad de registro de sucesos a ese entorno, y luego realizar el análisis tras su uso (Stottler, 2000; Stottler et al., 2001a).

El principal problema de esta aproximación es que la respuesta del sistema educativo puede llegar demasiado tarde. Normalmente se considera que

el mejor momento para proporcionar información es en el instante justo de necesitarla: el aprendizaje se potencia al ser puesto en práctica inmediatamente. Al retrasar tanto el *feedback*, los beneficios pedagógicos podrían dañarse.

La otra opción es monitorizar la resolución, y proporcionar ayuda, pistas o consejos *durante* todo el episodio de aprendizaje. En ese caso, esta etapa se *mezcla* con las dos etapas siguientes de análisis de la solución y comunicación del resultado. Es aquí donde cobran sentido los agentes pedagógicos de la sección 2.10, con sus problemas adicionales sobre cuando y cómo interrumpir que discutiremos más adelante.

Si queremos mantener nuestra similitud entre los sistemas educativos y los videojuegos, la disponibilidad de un entorno de aprendizaje es la *única opción posible*. Dado que estamos equiparando un ejercicio con un nivel del juego, el *entorno de aprendizaje* mencionado previamente será precisamente el propio videojuego. De hecho, desde el punto de vista de la implementación de la parte lúdica, esta fase será la más exigente.

La forma en la que la parte de videojuego y de aplicación educativa se entrelacen quedará patente aquí. En función de cómo se haya realizado la integración, quedará más o menos visible que el videojuego tiene carga pedagógica. Si el sistema debe ser capaz de proporcionar ayuda automáticamente, será necesario que durante la fase de diseño se haya pensado en cómo integrarla dentro del contexto del juego para que no resulte demasiado forzado. Ejemplos como PDA o personas de mayor rango en la jerarquía que dan órdenes son ejemplos claros en los tutoriales y niveles de muchos videojuegos (Splinter Cell, Half Life), por lo que podrían aprovecharse aquí. Otra opción es el uso de personajes auxiliares que acompañan al protagonista, cosa que también aparece en algunos videojuegos comerciales (Principe de Persia, Rayman 3) y que encaja perfectamente con la idea de los agentes pedagógicos.

4.7. Análisis de la solución

Para poder decidir si el estudiante está o no resolviendo correctamente el ejercicio, es necesario que el sistema *conozca la solución*. Al igual que en la creación del ejercicio, es responsabilidad del conocimiento del dominio ser capaz de proporcionarla, de modo que exista un método de comparar la labor del estudiante con la opción correcta para poder realizar el seguimiento correspondiente.

En la sección 4.5 decíamos que hay dos opciones principalmente para la elección de los ejercicios: *generación* al vuelo mediante un sistema experto, o *recuperación* de una base de ejercicios previamente creada y recopilada por un experto en el dominio. En lo que se refiere a la parte de resolución también disponemos principalmente de esas dos opciones.

Generación	Resolución	Tipo de tutoría
Sistema	Estudiante	Enseñanza o test
Estudiante	Sistema	Resolución para el estudiante
Sistema	Sistema	Demostración decidida por el sistema

Tabla 4.3: Combinaciones entre generación y resolución del ejercicio

Cuando en la parte de elección del ejercicio disponemos de un modelo que *genera* ejercicios, lo normal será utilizar también un sistema experto para resolverlos. Al fin y al cabo, si es posible crear el conocimiento necesario para *inventarse ejercicios coherentes* es muy probable que también resulte posible crearlo para *resolverlos*. Es lo que ocurre, por ejemplo en el tutor de C++ de Kumar (2002).

La semejanza entre el método de elección y de resolución del ejercicio no es sin embargo tan simétrica si en lugar de generar los ejercicios se seleccionan de una base de casos. En cierto modo, conseguir la posibilidad de *creación* de problemas es bastante difícil, pero en lo que se refiere a la *resolución* nos encontramos en un dominio mucho más habitual dentro del campo de la Inteligencia Artificial. Esto significa que es posible que no seamos capaces de construir un modelo para la generación automática de ejercicios pero que, aun así, podamos crear un sistema experto para su resolución. Es la aproximación seguida, por ejemplo, por Gertner y VanLehn (2000) en su sistema Andes para enseñar física o por Weber y Brusilovsky (2001) para enseñar LISP. Aunque los ejercicios son creados por un experto del dominio (siguiendo la aproximación CBR), luego disponen de un *modelo* que es capaz de resolver esos ejercicios automáticamente.

Una ventaja adicional de disponer de uno de tales sistemas expertos es que el sistema es capaz de resolver ejercicios *propuestos por el alumno*. Dependiendo del caso, puede resultar una buena medida si el sistema resulta ser incapaz de poner ejemplos clarificadores a las dudas del estudiante y éste prefiere plantear él mismo un caso cuya resolución resulte reveladora para sus dudas. Esta característica la tiene, por ejemplo, el sistema CPU City (Lester et al., 1999). Kumar (2002) resume las tres combinaciones interesantes en el cruce de quién genera y quién resuelve el ejercicio y su aplicación, que se resumen en la tabla 4.3.

No obstante, en dominios complejos, la creación de un sistema experto capaz de resolver *cualquier* ejercicio puede seguir siendo muy compleja. Cuando se dispone de una base de ejercicios, una alternativa muy natural es que el propio ejercicio *guarde su solución*. Aunque *la recuperación* (segunda etapa) siga realizándose en base a los *conceptos* a practicar, ésta extrae automáticamente ambas partes: enunciado y solución. El primero es planteado al estudiante, y el segundo se mantiene en el módulo del dominio como *ejerci-*

cio actual para tener la respuesta correcta con la que comparar los pasos del alumno. Esta es la aproximación utilizada, entre otros, por Stottler (2000) o Lewis Johnson et al. (2003).

Este modo de proceder tiene dos implicaciones. La primera es que la fase de adaptación (*reuse* en la terminología tradicional de *CBR*) que pudiera realizarse para ajustar el ejercicio recuperado a los conceptos que hay que practicar deberá también realizarse *sobre la solución*. Debido a la habitual dificultad de la fase de adaptación, la necesidad de retocar ambas partes añade un coste adicional que sólo suele ser posible si se utiliza KI-CBR.

La segunda es que el coste de autoría crece mucho. Aparte de la necesidad de crear los propios ejercicios (que viene implícita al seguir la aproximación CBR), los tutores se ven obligados también a dar la solución. Ya hemos dicho que es necesario que la base de ejercicios esté muy poblada para que el alumno tenga la sensación de que "nunca se termina"; forzar a los expertos del dominio a tener que especificar, una por una, cada una de las soluciones supone una carga de trabajo muy grande y, además, peligrosamente propensa a errores. La introducción en el sistema de un ejercicio *mal corregido* puede suponer una degradación en el aprendizaje por parte del estudiante bastante significativa. Incluso en el caso de que éste fuera capaz de darse cuenta del error, la pérdida de confianza en el sistema puede desmoronar todo el proceso.

Existe, sin embargo, una última opción que, creemos, no ha sido explorada. Dado que, al fin y al cabo, los sistemas de razonamiento basado en casos se utilizan para *resolver problemas*, un sistema de tutoría podría hacer uso de un sistema CBR *independiente* para la resolución de los ejercicios. En lugar de esperar que cada ejercicio recuperado en la segunda fase disponga de su propia solución, es posible utilizar una base de casos que relacione *enunciados* con su *solución*. Aunque en principio esto pueda no parecer ningún tipo de avance (al fin y al cabo necesitaremos seguir introduciendo manualmente las soluciones), dependiendo del tipo de dominio y la capacidad de adaptación de este segundo sistema CBR, podríamos *ahorrarnos muchas soluciones* de los ejercicios.

En el capítulo 6 ejemplificaremos esta idea. No obstante, para terminar de convencer aquí de sus ventajas, es necesario recordar que durante la segunda etapa de elección del ejercicio la consulta a la base de casos se realiza sobre los *conceptos* que se quieren practicar. Si se incluye adaptación en este sistema CBR, será para recuperar por conceptos (del dominio) parecidos, y retocar el enunciado eliminando conceptos que no quieren practicarse o "desplazando" otros parecidos para acercarlos a los que sí deben ponerse en práctica.

Si esta consulta en la segunda etapa recupera también la solución, la adaptación que se le aplicará será también en función de esos conceptos, para retocar aquellas partes relacionadas con el enunciado que se han visto alteradas.

Ahora bien, si consideramos el *espacio* de todos los ejercicios del domi-

nio recuperables, muchos de ellos tendrán soluciones relativamente similares. Sin embargo, si seguimos la primera aproximación, en la base de casos que relaciona conceptos con ejercicios, *todos ellos* necesitarán disponer de su correspondiente solución. Con nuestra idea de mantener dos sistemas CBR *independientes*, podemos introducir únicamente los pares enunciado-solución *prototípicos*, y desarrollar una adaptación específica para ellos, sin el yugo de los conceptos del dominio que pone en práctica cada enunciado, tal y como ocurría antes. Además, en aquellos dominios en los que un ejercicio puede tener varias soluciones válidas, este sistema escala mucho mejor si se compara con la alternativa de tener un único sistema CBR.

Incluso aunque esta argumentación pueda seguir siendo, para algunos, algo discutible, proporciona además una ventaja adicional que resultaría inviable en el caso de realizar la recuperación de la solución directamente en la segunda etapa. Para que un tutor inteligente sea realmente útil no sólo es suficiente con que conozca la solución *correcta*: también necesita explicar por qué las soluciones incorrectas lo son. Si sólo mantenemos un sistema CBR que nos recupera simultáneamente el ejercicio y su solución, tendríamos que mantener, además, todas las equivocaciones típicas de ese ejercicio para ser capaces de proporcionar información adecuada cuando el estudiante se confunda. Salvo que el número de posibilidades sea pequeño, esto resulta completamente inviable por culpa de la explosión combinatoria.

Sin embargo, si utilizamos la aproximación de un segundo sistema CBR para la resolución, podemos mantener dentro de la base de casos *soluciones inválidas* sin apenas ningún esfuerzo adicional. Si cada una de ellas es anotada por un experto explicando la razón del error, podemos disponer de respuestas acertadas en los errores más habituales de los alumnos. En realidad, esto puede considerarse un retroceso, de nuevo, hacia los sistemas basados en marcos en los que había que controlar todas las posibles equivocaciones. Esta crítica es, hasta cierto punto, acertada; la diferencia es que la capacidad de adaptación del sistema CBR, y de reutilización entre diferentes ejercicios alivia mucho el esfuerzo de autoría, a la vez que aleja el fantasma de la creación de un sistema experto, algo que podría no ser viable en dominios complejos.

Un último punto interesante es que la base de casos *de error* puede poblarse de forma iterativa. Si el sistema se ve incapaz de dar una explicación justificada al error de un estudiante, la razón será que o bien la base de casos es escasa, o bien el proceso de adaptación es pobre. En cualquier caso, el sistema puede registrar este hecho para que durante la fase de mantenimiento se detecten esas carencias y puedan ser inmediatamente resueltas. Aunque este tipo de registros también pueden realizarse en el caso de utilizar sistemas expertos, gracias al funcionamiento episódico del CBR resulta muy sencillo solventar las deficiencias (basta con añadir un nuevo caso), algo que no es tan claro en la otra aproximación.

Naturalmente, si se utiliza un sistema experto en lugar de casos para resolver los ejercicios, también debería poder proporcionarse información específica para cada tipo de error. Esto exige *transparencia* en dicho sistema experto, para poder seguir la pista de los pasos correctos, encontrar dónde se ha desviado el alumno y poder así intentar dar una justificación. Por desgracia, el campo de la Inteligencia Artificial tiene mucha experiencia en la resolución de problemas, pero no tanto en el control de los posibles errores, por lo que esta alternativa no siempre es sencilla.

En ese caso necesitaremos de nuevo el *módulo del usuario* que, una vez averiguado el error, le dé una explicación. Es decir, a partir del *fenotipo* del error debe generar el *genotipo* (Hollnagel, 1991) o, dicho de otra forma, las creencias, objetivos y conocimiento del dominio que el estudiante tiene y que le han llevado a dar el paso en falso. Esta labor es responsabilidad del que denominábamos módulo del estudiante *a corto plazo*. La forma en la que se realice dependerá de cada sistema particular y, especialmente, del modo en el que podamos obtener información del módulo cognitivo. Por ejemplo, el sistema Andes para enseñar física maneja toda la incertidumbre de esta etapa haciendo uso de redes bayesianas (Conati et al., 2002).

En principio, este *módulo del usuario* no sería estrictamente necesario en la aproximación CBR, dado que habrá sido el experto quien haya tenido que prever los posibles fallos y sus causas, para añadirlas como respuestas inválidas dentro de la base de casos. De nuevo, esto se debe a que en general la alternativa CBR es más *pobre en conocimiento* que un sistema experto equivalente. Como siempre, la elección de una u otra opción dependerá fuertemente del *dominio*: si es complejo, la posibilidad de disponer de un modelo completo puede ser, sencillamente, imposible. No obstante, no se puede descartar la necesidad de un modelo del estudiante a corto plazo ni siquiera en el caso de utilizar una aproximación CBR. En ocasiones (especialmente durante el análisis de *soluciones parciales*), podrían aparecer varias razones posibles para un error, por lo que la información sobre el alumno sería útil para *desambiguar*.

Hasta ahora, se ha estado dejando de lado el hecho de que nuestra idea es integrar el sistema de tutoría inteligente dentro de un videojuego. Dado que cada ejercicio es un *nivel* del juego, la monitorización del estudiante para averiguar qué está haciendo mal y qué está haciendo bien debe eliminar el "ruido" provocado por la parte lúdica dentro del entorno de aprendizaje. Así, es necesario realizar un filtrado para proporcionar a la parte educativa únicamente los pasos importantes, y no avasallarla con gran cantidad de información no importante.

El verdadero problema aquí es dónde colocar el límite, es decir qué se debe considerar parte relacionada con la parte educativa y qué con la parte lúdica. En realidad, la ilusión de inteligencia y ayuda que puede crear el sistema proporcionando información durante la resolución de un problema

se desbaratará si no es capaz de suministrar información básica sobre el entorno, o sobre las reglas de la parte lúdica.

Asumiendo ese riesgo, la parte educativa podría quedar a la escucha sólo de los eventos importantes relacionados con el dominio que se enseña, por ejemplo "el usuario ha pulsado una válvula" o "el estudiante ha entrado en la sala de mandos". Pero si la ayuda es proporcionada a través de un agente pedagógico, que *habita* en el entorno de aprendizaje, éste tendrá que ser consciente del espacio en el que reside, controlar dónde está el estudiante para seguirle, para mirarle cuando le hable o para llamarle la atención si no le "atiende" cuando le da su ayuda (Johnson et al., 1998; Rickel y Johnson, 1999). Esto significa que incluso las cosas más insignificantes podrían resultar de interés para la parte pedagógica del sistema, exigiendo una colaboración mucho más estrecha entre ambas componentes.

4.8. Comunicación del resultado

Gracias a la etapa anterior el sistema puede hacerse una idea sobre la situación de la resolución del ejercicio, ya sea una solución parcial intermedia o la solución final (con, quizá, todos sus pasos).

En una clase tradicional, el profesor proporcionará, en este punto, información específica para el alumno en función de las deducciones sobre las causas de los errores. Como ya dijimos, debido a la enseñanza en grupo, la posibilidad de proporcionar explicaciones adaptadas a cada alumno resulta complicada por la falta de tiempo. El resultado suele consistir en una serie de avisos sobre los fallos que más se han cometido dentro del grupo, pero sin particularizar cada caso concreto.

Sin embargo, cuando es un ordenador el que está haciendo las veces de tutor, es posible construir explicaciones adaptadas a cada usuario. En realidad, habrá sido responsabilidad de la fase anterior extraer las conclusiones necesarias sobre las *causas* que han llevado al error (usando el *modelo del estudiante*). En esta etapa es necesario plantearse *cómo* explicárselas.

Además, ¿cuando se proporciona esta información? En los sistemas que no constan de un entorno de aprendizaje o que no lo tienen integrado con la cuarta etapa (de monitorización), tendremos que esperar hasta el final del ejercicio para hacerlo. Al fin y al cabo, esta etapa de generación de explicaciones está esclavizada a la etapa de análisis de la solución, por lo que no podrá entrar en acción hasta que dicho análisis no haya proclamado un veredicto. Este es el caso, por ejemplo, del sistema presentado por Stottler (2000) ya mencionado, en el que se adaptó un simulador para que generara información de registro, pero el análisis de los pasos realizados por el estudiante se retrasó hasta la finalización del ejercicio.

Esta alternativa de posponer el suministro de información también está disponible incluso en el caso en el que el sistema tenga la posibilidad de

analizar la solución *durante* la resolución, es decir cuando la etapa 3 y 4 están fusionadas. No obstante, en ese caso normalmente se considera mucho más interesante proporcionar ayuda durante todo el proceso. En concreto, el sistema podría ser *reactivo* quedando a la espera de que el estudiante pida ayuda de manera explícita (Nogry et al., 2004) o *proactivo* proporcionándola cuando se detecta una necesidad (Gertner y VanLehn, 2000; Weber y Brusilovsky, 2001). En este último caso, es interesante suministrar información *contextualizada* a la situación actual, en lugar de ayuda genérica más o menos aplicable a ésta. Por ejemplo, Stone y Lester (1996) disponen de frases específicas para las diferentes situaciones que consiguen esta adaptación al contexto.

Aquí surge de nuevo la pregunta, esbozada en la sección anterior, sobre quién proporcionará esa ayuda. Para que la integración sea fluida, es mejor integrar en el entorno de aprendizaje un agente pedagógico. Ya comentamos que éste requerirá información sobre el espacio virtual en el que habita para poder desenvolverse en él, así como capacidad para la *generación superficial* de la información que se desea transmitir y proporcionada por la etapa anterior. Todo esto entra, en nuestra opinión, en la categoría del *conocimiento de interfaz* de la organización clásica de los tutores inteligentes.

Incluso en el caso de que el sistema pueda proporcionar información durante todo el proceso de resolución, una cuestión a plantearse es si realmente *es adecuado* hacerlo en un determinado momento. Dependiendo del estilo de aprendizaje, interrumpir continuamente al estudiante puede coartar su libertad y perjudicar más de lo que beneficia (basta con recordar las dudas que Skinner tenía sobre el modo de educar a su hija que se mencionaron en la página 66). Esta decisión sobre si intervenir o no es tomada por el módulo pedagógico.

Cuando un sistema educativo está integrado dentro de un videojuego, la elección sobre si intervenir o no es todavía más delicada, porque no sólo afecta a la parte de aprendizaje, sino también a la de entretenimiento, con la correspondiente repercusión en la motivación. Normalmente los videojuegos permiten *exploración libre*, y prácticamente todo el aprendizaje se consigue a través del descubrimiento y de la respuesta del sistema a cada acción. Si al añadir la capacidad de tutoría esta sensación de libertad se pierde, una de las partes divertidas del juego (la *ineficiencia* de hacer lo que a uno le venga en gana) desaparece. Es necesario un equilibrio entre enseñanza (eficiente) y libertad (entretenida pero lenta).

No obstante, gracias a la riqueza de los videojuegos, podemos proporcionar información importante al estudiante *de manera sutil* en lugar de interviniendo drásticamente y deteniendo el fluir de los acontecimientos. En realidad, los agentes pedagógicos ya disponían de esta posibilidad a través de comunicación no verbal. También podemos mostrar pistas al estilo del juego infantil *frío-caliente*, algo que hace el sistema de tutoría Andes con

colores, o *Food Force* (el juego para concienciar sobre el hambre en el mundo, figura 4.4b) en una de sus fases. Otro punto de *realimentación* inmediata es la puntuación. El lenguaje de los videojuegos cuenta con esta herramienta desde tiempos inmemoriales: si lo haces bien te doy puntos, y si no, no te los doy. Aprovechando este ingrediente habitual, podemos indicar de manera rápida y no intrusiva (para no perder la sensación de estar dentro de un juego) si el estudiante avanza correctamente o no.

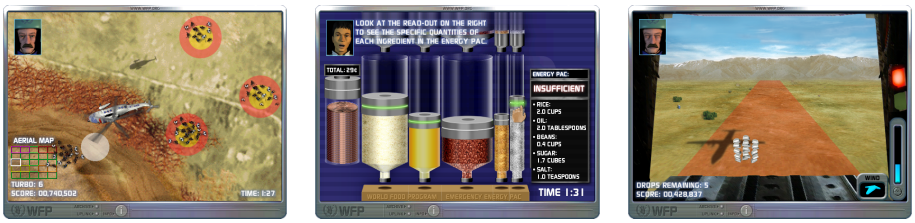
4.9. Integración de los ejercicios y los niveles del juego

A lo largo de las secciones anteriores hemos hecho un recorrido sobre el modo en el que los diferentes tipos de conocimiento de los tutores inteligentes se utilizan en un sistema educativo basado en ejercicios. La conclusión más importante es que necesitamos algún modo de disponer de *ejercicios* y sus *soluciones*, ya sean creados automáticamente (con sistemas expertos) o manualmente (mediante casos con más o menos conocimiento generalista). En dominios con un mínimo de complejidad, resulta poco probable conseguir un modo de crear los ejercicios de manera procedimental, por lo que habitualmente se dispondrá de una batería de ejercicios aptos para diferentes situaciones.

En lo que se refiere a los videojuegos basados en niveles, también existen dos alternativas similares. La primera es la generación procedimental de dichos niveles, en los que la principal labor cae en el lado de la programación. La segunda es la creación manual, donde la carga principal se desplaza a los diseñadores que han de construir todos los niveles más o menos desde el principio. A día de hoy, es esta última la alternativa más utilizada. Los equipos de desarrollo quieren tener bajo control la evolución del juego, y prefieren dejar en tiempo de ejecución el menor número posible de aspectos al azar para que no haya disgustos.

Al fusionar sistemas educativos y videojuegos en un intento de aumentar la motivación de los primeros, el desarrollo se convierte en una mezcla que debe atender a ambos aspectos. En función del tipo de integración decidida en el diseño, surgen modelos de implementación diferentes, dependiendo de lo enraizados que estén los ejercicios y el resto de componentes pedagógicos en los lúdicos.

Dado que lo más habitual es tener los ejercicios (en el lado educativo) y los niveles (en el lado lúdico) hechos a mano, la alternativa más inmediata que surge es crear un sistema educativo en el que ambos continúen siendo creados manualmente. En ese caso, el diseñador de cada nivel es a la vez el creador del ejercicio que integra. Esto exige que una misma persona sea experta en diseño de videojuegos y en el dominio que se enseña. Por otro lado, ya



(a) Estimación del número de afectados

(b) Optimización de las raciones alimenticias

(c) Entrega por avión

Figura 4.4: Food Force (`http://www.food-force.com/`)

dijimos que si se sigue la aproximación basada en casos para los ejercicios, es posible, si se cuenta con el conocimiento necesario, retocarlos para ajustarlos mejor a la situación particular. Si los ejercicios están integrados en los niveles del juego, es mucho más complicado realizar adaptación porque es necesario realizarlo en los dos aspectos: ejercicio y juego.

No obstante, esta aproximación es viable en algunas ocasiones. Si el dominio que se desea enseñar es lo bastante pequeño, podría *enseñarse en un único nivel*, en cuyo caso es posible dedicar esfuerzo en construirlo correctamente. Naturalmente, de hacerse así nos alejaríamos de la enseñanza basada en ejercicios. Esta aproximación es la seguida por Moreno Ger et al. (2005) y Martínez Ortiz et al. (2006) al construir *objetos de aprendizaje* como juegos de aventura.

También esta alternativa es la utilizada por *Food Force*, en el que aparecen 6 niveles totalmente independientes. Cada uno de ellos tiene un objetivo completamente diferente, aunque siempre persiguen concienciar sobre los problemas de las organizaciones no gubernamentales en las crisis humanitarias. Por ejemplo, el primero intenta mostrar la dificultad de estimar el número de afectados (figura 4.4a), el segundo el modo de optimizar el coste de las raciones alimentarias y su capacidad nutritiva (figura 4.4b) y el tercero las complicaciones de hacerlas llegar por avión (figura 4.4c).

Con esta aproximación, las necesidades de autoría se acercan más a las de los videojuegos que a las de los sistemas educativos. En función del género del juego y el modo de implementación elegidos se utilizarán unas herramientas u otras. Por ejemplo, en el caso de los minijuegos de aventura en los objetos de aprendizaje, sus autores hacen uso de un lenguaje de creación de historias propio que es interpretado por un *applet* de Java integrado en el LMS. Por su parte, *Food Force* está desarrollado con Macromedia Director, por lo que la autoría se enfoca principalmente a componentes visuales, sonidos y de programación.

De todas formas, en ninguno de los dos casos, se sigue una aproximación educativa basada en ejercicios en el que éstos son cada vez más difíciles

dentro de un dominio complejo. Si quisiéramos utilizar de verdad esta aproximación, y aun así mantener la creación manual de ejercicios y niveles para que estuvieran integrados, sufriríamos un incremento en los requisitos de autoría difícil de soportar. Si, además, nos empeñamos en proporcionar ayuda contextualizada, adaptación al estudiante y el resto de aspectos determinantes de una tutoría inteligente, la explosión combinatoria terminaría siendo imposible de saciar.

Por tanto, en dominios un poco más grandes en los que queramos mantener la adaptación al usuario y seguir de verdad una aproximación basada en ejercicios, es preferible mantener una separación entre ambas partes. La componente educativa del sistema mantiene un funcionamiento orientado a los ejercicios, controlando lo que sabe el estudiante, planteando la mejor actividad para el siguiente episodio de aprendizaje y determinando las razones de las equivocaciones del estudiante. En este caso, es indiferente si los ejercicios son creados al vuelo por un sistema experto, o se extraen de una batería siguiendo la aproximación basada en casos.

Luego será necesario algún modo de enlace entre los ejercicios y el nivel. Esta relación se crea de manera procedimental pues disponer de antemano de todos los posibles niveles para cualquier posible ejercicio nos haría caer en los problemas de autoría descritos antes.

Dependiendo del dominio, la creación al vuelo del nivel de juego puede ser completa, o basada en contenido auxiliar que se *combina* para *configurarlo* en función de las necesidades del ejercicio. Esto de nuevo hace que la carga se reparta un poco, aliviando los problemas de la creación completa del ejercicio mediante código al poder hacer uso de, por ejemplo, modelos y mapas prototípicos que luego son retocados. En cualquier caso, es necesario añadir *variedad* en el mapeo entre los ejercicios y los niveles, para evitar la rutina y tratar así de mantener la diversión gracias a la impredecibilidad.

La separación entre ambas partes permite también hacer uso de herramientas específicas para cada una. Dependiendo de si en la componente educativa se utiliza un sistema experto, un sistema basado en casos puro o nuestra propuesta de KI-CBR serán necesarias unas u otras herramientas. En cualquier caso, la ventaja principal es que seguramente podamos hacer uso de algunas herramientas generalistas heredadas de la Inteligencia Artificial. Esto no sólo incluye aquellas pensadas para la definición del propio conocimiento, sino también en tiempo de ejecución como los motores de inferencia.

El punto negativo es que resulta mucho más complicado apoyarse en videojuegos comerciales para modificarlos, porque es difícil integrar en ellos este tipo de herramientas. Por tanto, es necesario también una carga de autoría en la parte lúdica, aunque de nuevo es posible utilizar herramientas generalistas, como entornos de diseño gráfico 2D y 3D. Además, a nivel de implementación podremos utilizar *motores* de juego que descarguen un poco

el trabajo. De todas formas, seguirá siendo necesaria la programación de los aspectos procedimentales que nos permitan unir los ejercicios con los niveles.

Conclusiones

Los videojuegos educativos del capítulo anterior sufren varios problemas cuando se intenta extrapolar sus ideas a dominios complejos. Quizá el principal es la *carencia de tutoría*. Para áreas sencillas, es posible confiar en el aprendizaje por descubrimiento, pero cuando la materia enseñada adquiere una dificultad mayor, surge la clara necesidad de algún tipo de guía. Vimos que ese papel podía ser llevado a cabo por el profesor, pero si queremos perseguir el sueño de la educación personalizada es necesario que sea el propio ordenador el que haga de guía.

La investigación en tutores inteligentes lleva mucho tiempo persiguiendo este objetivo, con un cierto éxito. Por tanto, para poder aprovechar ese camino ya recorrido e integrar sus ideas dentro de los videojuegos educativos, es necesario encontrar algún tipo de paralelismo entre el funcionamiento de un sistema educativo y un videojuego. Así, conseguiremos incluir las ideas de los primeros en los segundos de una manera ordenada e, idealmente, reutilizable.

En este capítulo hemos descrito nuestra propuesta de unión, consistente en asociar a cada *nivel del juego* un *ejercicio*. Con esta fusión, los ciclos de funcionamiento de un tutor inteligente y de un juego resultan de una similitud sorprendente, abriendo la puerta a una integración controlable.

No tenemos noticia de videojuegos educativos que pongan en práctica estas ideas. Existen algunos intentos, pero habitualmente hacen uso de una fusión total entre el ejercicio y el nivel en lo referente a implementación. Esto supone un gran problema de autoría, tanto por los requisitos de conocimiento del diseñador como por la dificultad de modificar al vuelo los ejercicios, obligando a la creación manual de una gran cantidad de ellos para no dañar la adaptación al estudiante. En la práctica, termina saliendo perdiendo esta adaptación, y la elección de los niveles resulta lineal (o incluso aleatoria).

Nuestra propuesta es mantener separados los ejercicios y los niveles, creando estos últimos de manera procedimental a partir de los primeros, que son los que dirigen, en cierto modo, la evolución del juego. La parte de tutoría funciona de manera relativamente independiente de la parte lúdica, lo que permite separar el trabajo de autoría entre ambas. Esto facilita la inclusión de inteligencia que guíe al alumno, pues resulta posible echar mano del conocimiento que la inteligencia artificial y, en particular, el mundo de los tutores inteligentes ha ido acaparando a lo largo de los años. Los aspectos lúdicos quedan de esta forma "por debajo", sin interferir en exceso en la parte educativa pero, aun así, es posible utilizar *integración intrínseca* si la unión entre ambas partes de realiza con delicadeza.

La figura 4.5 resume estas ideas. En la parte superior se muestran los ele-

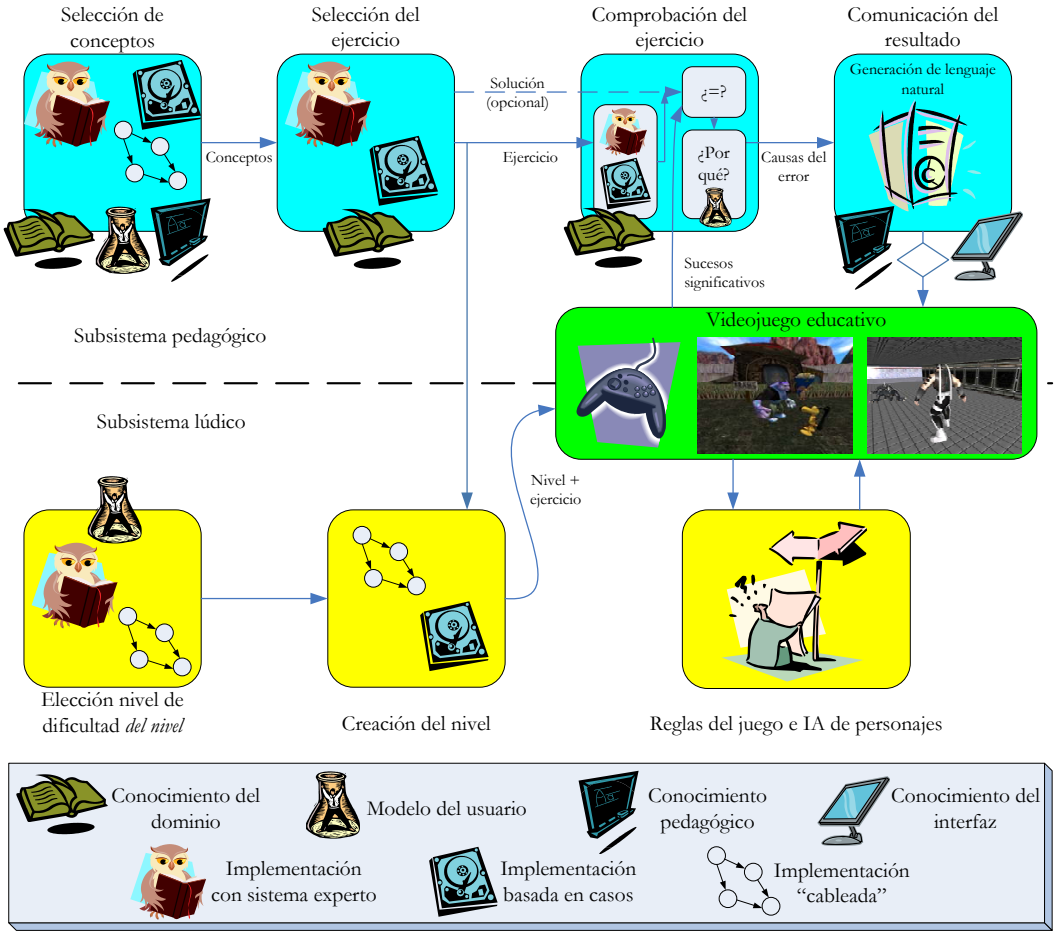


Figura 4.5: Esquema lógico del modelo para videojuegos educativos

mentos relacionados con la componente pedagógica del sistema, y la inferior muestra aquellos pertenecientes a la parte lúdica. Ambos quedan fusionados en el videojuego educativo, representado en la zona central, al incrustarse un ejercicio en un nivel de juego.

Siguiendo este modelo, el programa comienza seleccionando los siguientes conceptos a practicar. La figura muestra (con los iconos situados en la parte inferior del bloque que representa esta etapa) que para realizar este paso se utiliza conocimiento del dominio, pedagógico, y el modelo del usuario, tal y como resumía la tabla 4.1. La implementación dependerá en gran medida del modo en el que dicho conocimiento esté almacenado. Como quedó dicho en la sección 4.4, puede utilizarse un sistema experto, seguirse una aproximación basada en casos, o incluso una implementación totalmente decidida de antemano en el que el sistema no realiza ningún tipo de "razonamiento".

Los conceptos seleccionados se pasan a la siguiente etapa con la que se selecciona un ejercicio. De nuevo en función del modo en el que quede representado el conocimiento del dominio, podrá utilizarse un sistema experto para generarlo al vuelo, o una base de ejercicios previamente alimentada por expertos.

De manera paralela, el programa habrá decidido la dificultad del nivel (parte lúdica) para ajustarla también al perfil del estudiante. Con el ejercicio y esta información se *crea un nivel* de juego. En este punto es donde quedan fusionados ambos mundos, y donde el *diseño* del juego se hace patente. Las posibles formas de realizar esto son amplias, y entroncan con los problemas habituales existentes en el desarrollo de videojuegos. Aunque se han dado ciertas ideas en la sección 4.9, el tema tiene gran amplitud y queda fuera de este trabajo. Es tratado con detalle por Gómez Martín (2007).

El nivel de juego, con el ejercicio integrado, es lanzado en el entorno del videojuego, mostrado en la parte central. Durante su ejecución, la parte lúdica sigue en ejecución llevando a cabo la lógica del juego y la Inteligencia Artificial de los personajes (*bots*) auxiliares. Por su parte, las dos últimas etapas del ciclo de nuestro modelo también se mantienen en marcha.

En concreto, el entorno virtual *filtra* las acciones y proporciona al componente pedagógico únicamente aquellas que tengan importancia en la parte educativa. Si durante la selección del ejercicio no se proporcionó también su solución, el sistema la crea ahora (o bien con reglas o con casos) y la compara con lo que el estudiante está haciendo. Si hay algún desajuste, utilizamos el modelo del estudiante para averiguar la causa del problema. Ésta es proporcionada a la última etapa, o bien mediante texto enlatado o mediante algún otro tipo de representación más amigable para el ordenador.

En la última etapa decidimos, a partir del conocimiento pedagógico, si inyectar o no de vuelta al entorno la explicación, consejo o indicación. Esto permite que el sistema controle en qué momentos es preferible interrumpir. Del mismo modo, usando conocimiento de interfaz se plantea la mejor forma de hacer llegar al estudiante la información a través del entorno.

Como ya habíamos anticipado en la introducción del capítulo, el punto de unión entre nivel y ejercicio sigue dejando alrededor muchos puntos sin fijar, que proporcionan una gran flexibilidad al modelo. En concreto, no se fuerza a ningún modo concreto de representar el conocimiento, por lo que, del mismo modo, no se exige un tipo específico de implementación. La figura deja patente este hecho, al mostrar en cada componente del sistema los iconos tanto de la implementación mediante un sistema experto como utilizando casos en aquellos puntos donde ambas podrían tener sentido.

Es necesario hacer notar que la elección de una aproximación en una etapa no necesariamente ata a las demás. Por ejemplo, podemos utilizar una implementación cableada en la selección de conceptos, una base de ejercicios para seleccionarlos, y un sistema experto para resolverlos. Esto demuestra

la relativa independencia entre cada una de las etapas, que podrían desarrollarse de manera razonablemente separada.

Naturalmente, en este caso la carga de autoría de conocimiento puede crecer más allá de lo deseable. Nosotros preferimos una alternativa que se sitúa a medio camino entre los sistemas expertos y los basados en casos. El conocido como razonamiento basado en casos rico en conocimiento (KI-CBR) nos permite así quedarnos con lo mejor de ambos mundos. Los casos abren la puerta a dominios más complejos en los que resulta imposible crear un sistema experto que los modelice, mientras que el conocimiento declarativo con el que se enriquecen los casos ahorra parte de la carga de autoría al poder reutilizar algunos aspectos que son tratados automáticamente por el sistema.

Además, podemos utilizar bases de casos diferentes y *adaptadas* a cada una de las etapas que, aun así, *compartan* el conocimiento declarativo del dominio. Esto ahorra trabajo de autoría, dado que el esfuerzo necesario para crear este conocimiento genérico se *amortiza* en los diferentes subsistemas, a la vez que permite mantenerlos unidos. Otra ventaja de la disponibilidad de este conocimiento adicional es que facilita la etapa de adaptación (*reuse*) del CBR, pudiendo así crear la ilusión de una mayor base de ejercicios que se ajusta mejor a los requisitos de los alumnos.

En el próximo capítulo...

Los dos capítulos anteriores han servido de antesala y preparación para la unión de ambos mundos (educación y juegos) planteada en éste. La fusión de un ejercicio y un nivel consigue un engranaje perfecto entre los ciclos de funcionamiento de un videojuego y de un sistema educativo. Alrededor, sin embargo, se dejan abiertos varios aspectos de implementación en los que los desarrolladores deben tomar decisiones particulares que se ajusten bien a su dominio.

En los dos capítulos siguientes veremos dos encarnaciones diferentes de estas ideas, en las que se toman diferentes caminos para la implementación de cada una de las etapas. Aunque en ambos casos el objetivo del sistema es el mismo (enseñar de manera intuitiva cómo traducir código fuente en código objeto), las decisiones de implementación tomadas son bastante diferentes. Esto demostrará por un lado la generalidad del modelo planteado en este capítulo, pues no sólo no fuerza a un determinado modo de representación de conocimiento, sino que ni siquiera éste es del todo forzado por el dominio. Por otro lado, los aspectos lúdicos de cada uno de los sistemas son también bastante diferentes. Esto también deja patente que, al igual que el modelo no fuerza elecciones concretas en la parte pedagógica, tampoco ata a los diseñadores a un género de videojuego determinado, o a un tipo de integración entre ambas componentes más allá de la dualidad ejercicio–nivel.

En concreto, el capítulo siguiente describe un juego de aventura en el que se utilizan casos con muy poco conocimiento adicional para soportar la enseñanza. Esto sirve de antesala para demostrar las ventajas del KI-CBR que se realiza en el capítulo 6 a través de un videojuego de acción.

Capítulo 5

JV^{2}M: Realización del modelo basada en casos

– Paréceme, ¡oh Anselmo!, que tienes tú ahora el ingenio como [...] los moros, a los [...] que les han de traer ejemplos palpables, fáciles, intelegibles, demonstrativos, indubitables.

Don Quijote de la Mancha, Miguel de Cervantes

Resumen: En este capítulo se describe una instanciación del modelo de videojuegos educativos que se planteó en el capítulo anterior. Se particularizan los diferentes puntos que dicho modelo dejaba abiertos utilizando una aproximación basada en casos tradicional. Esto pone a prueba la validez de esas ideas en un marco pedagógico específico.

5.1. Introducción

En el capítulo anterior se propuso un modo de fusionar las ideas de las aplicaciones de apoyo a la enseñanza y los videojuegos, que fueron originalmente tratados en los capítulos 2 y 3 respectivamente. El punto principal de unión consiste en seguir una aproximación de enseñanza basada en ejercicios, e incrustar en un nivel del videojuego un ejercicio. Esto supone que la *resolución* de dichos ejercicios es siempre realizada a través del propio sistema, algo que no siempre ocurría en otras aplicaciones educativas.

Este modelo pone en contacto, por tanto, la componente lúdica y educativa en el nivel, pero deja abierto el modo de implementación del resto de subsistemas. En concreto, se planteó una división de la ejecución en diferentes fases, y se describieron las distintas formas de afrontar su desarrollo.

Para poner a prueba la validez de las ideas, en éste y en el capítulo siguiente planteamos dos alternativas de implementación para el mismo dominio. Sorprendentemente, aunque ambas siguen el mismo modelo y enseñan el mismo área, las instanciaciones particulares de las ideas del capítulo anterior son bastante diferentes. Naturalmente, en ambos casos se mantiene la relación entre un nivel y un ejercicio, punto central de la fusión planteada. Esto significa que la resolución del ejercicio (la que llamábamos tercera etapa del ciclo) se realiza irremediablemente a través del ordenador.

Sin embargo la forma en la que se realizan las otras cuatro etapas difiere. Algunos cambios son claramente apreciables por el usuario, debido al cambio de género de videojuego. Otros no lo son tanto, y afectan más al modo en el que la aplicación se ha desarrollado, y cómo se le ha añadido la capacidad de tutoría.

Este capítulo comienza describiendo el *dominio* que ambas aplicaciones intentan enseñar. Luego analiza cómo se ha fusionado en esta primera instanciación del modelo la parte pedagógica y lúdica, ajustándola a un género de videojuego bien conocido. Esto da una idea sobre el funcionamiento general de la aplicación tal y como es percibida por el usuario. Posteriormente se disecciona la forma en la que se realizan cada una de las cinco etapas de ejecución propuestas en el capítulo anterior. El capítulo termina describiendo el conocimiento que el sistema necesita para llevar a cabo su labor.

5.2. El dominio: traducción de código fuente a código objeto

Para ejemplificar las ideas expuestas en el capítulo 4 hemos realizado dos instanciaciones diferentes del modelo. Ambas están centradas en enseñar un mismo dominio, y ambas tienen como eje de unión principal la dualidad ejercicio–nivel del juego. A pesar de todo, los sistemas resultantes son bastante diferentes, tanto en el tipo de juego percibido por el estudiante como en el funcionamiento interno a nivel de la tutoría inteligente.

Antes de pasar a detallar en profundidad la primera de las instanciaciones planteadas, que denominamos JV^{2}M, es necesario detenerse en *el dominio* que pretendemos enseñar. Se escogió un dominio cercano ya conocido para evitar una dependencia con expertos externos. En concreto, queremos enseñar, de una manera *intuitiva* cómo traducir código fuente en código objeto. Estos conocimientos encajan en la asignatura troncal *Procesadores de Lenguaje*, impartida en cuarto de Ingeniería en Informática.

No pretendemos cubrir, ni mucho menos, el temario completo de dicha asignatura. En concreto, la intención *no* es proporcionar una base teórica y sólida sobre el funcionamiento de los compiladores, por lo que no se plantean conceptos tales como gramáticas, análisis léxico, sintáctico, o tabla de

símbolos. En lugar de eso, queremos despertar y agilizar el sentido común sobre la traducción, evitando los requisitos teóricos subyacentes, necesarios para la implementación.

Una vez elegido a grandes rasgos este dominio, es necesario particularizarlo para *un lenguaje de programación* concreto. En las clases regladas de Procesadores de Lenguaje es habitual hacer uso de lenguajes sencillos e inventados, que se van haciendo cada vez más complejos según se van incrementando las construcciones que se enseñan a compilar. Por ejemplo, se suele empezar con expresiones aritméticas sencillas, a las que se les añaden variables, tipos de datos, etcétera.

Esta progresión en la complejidad del lenguaje está respaldada por la necesidad de ir introduciendo de manera paulatina los conceptos teóricos subyacentes. Comenzar con las sencillas expresiones aritméticas permite eliminar *ruido*, y facilita la comprensión de la necesidad de separación entre las fases de análisis léxico y sintáctico. La inclusión de variables sirve para introducir la tabla de símbolos, mientras que las primeras estructuras de control (típicamente el `if`) permiten presentar las instrucciones de salto.

Dado que nuestra intención no es enseñar toda la base teórica subyacente, el uso de un lenguaje inventado que se va haciendo crecer progresivamente supone, en nuestro caso, más una desventaja que una ventaja. Queremos conseguir que los alumnos sean capaces de generar código objeto a partir del código fuente por sí mismos haciendo uso de la intuición, no de los artefactos que se encuentran incrustados en un compilador. Para eso necesitamos que *conozcan el lenguaje*, para evitar añadir un obstáculo adicional que deban aprender. Esto, de hecho, es una cuestión que es necesario recalcar: *no* queremos enseñar a programar, de ahí que prefiramos utilizar un lenguaje fuente lo más extendido posible para aumentar la probabilidad de que sea conocido de antemano.

Otra práctica habitual en los cursos de Procesadores de Lenguaje es utilizar, también, lenguajes *objeto* inventados a los que traducir el código fuente. Aunque las aproximaciones son variadas, no es extraño el uso de lenguajes basados en pila (muy cómodos para las expresiones aritméticas iniciales), que luego se van ampliando según evoluciona el curso.

En nuestro caso, creemos que forzar el uso de un lenguaje objeto arbitrario puede resultar crítico por carecer de interés. Por desgracia, las alternativas son, sin embargo, algo escasas. En el caso del lenguaje *fuente* es cuestión de escoger uno con suficiente penetración como para que sea conocido por el mayor número de estudiantes posible. Sin embargo, es muy poco común que los alumnos conozcan un lenguaje objeto con la suficiente soltura como para que no sea necesario enseñarlo también.

Ante este panorama, elegimos utilizar Java como lenguaje fuente. Consideramos que está lo suficientemente extendido como para que pueda ser conocido por la gran mayoría de nuestro público objetivo. Además, como la

sintaxis imita la de C++, se amplía todavía más el rango de personas que podrían entender sin demasiado esfuerzo el código que se muestre.

Una ventaja adicional del uso de Java es que la elección del lenguaje objeto es inmediata: el lenguaje interpretado por la JVM (*Java Virtual Machine*, Máquina virtual de Java). Si hubiéramos escogido un lenguaje como C++ o Pascal, el lenguaje objeto podría haber sido el ensamblador de cualquiera de las máquinas sobre las que estos lenguajes compilan. Sin embargo, en el caso de Java, los programas siempre se traducen al mismo conjunto de instrucciones "máquina", que luego son *interpretadas* por la JVM.

El uso de este lenguaje objeto (impuesto, en cierto modo, por la elección de Java como lenguaje fuente) tiene además varias ventajas adicionales:

- Las instrucciones son razonablemente sencillas. Comprender las instrucciones de un ensamblador real exige conocer la arquitectura de su procesador y sus modos de direccionamiento con un cierto detalle, y cada uno tiene sus propias peculiaridades. La JVM es una máquina basada en pila bastante sencilla que simplifica su curva de aprendizaje.

- El lenguaje objeto de la JVM es más interesante que el de un ensamblador inventado con motivos pedagógicos, dado que *existe* y se utiliza ampliamente alrededor del mundo.

- Tiene instrucciones que proporcionan un soporte nativo para la orientación a objetos. En condiciones normales (cuando se compila a máquinas reales), las características principales de la orientación a objetos (como la vinculación dinámica) requieren el uso en ejecución de construcciones complejas que el propio código compilado debe controlar. Dado que es directamente la JVM quien se encarga de todas ellas de manera nativa, es posible ahorrar muchos detalles en lo que se refiere a generación de código, y a pesar de eso poner a prueba los conocimientos sobre las repercusiones de la orientación a objetos.

Consideramos que la elección de Java es una decisión afortunada porque dado que *no* pretendemos explicar la base teórica del proceso de compilación, el sistema podría utilizarse incluso en cursos anteriores al de la asignatura de Procesadores de Lenguaje. Como se proporciona un aprendizaje intuitivo, encaja perfectamente en un curso de Java, para mostrar qué ocurre más allá del editor de código fuente.

5.3. Descripción de la JVM

Antes de adentrarnos en cómo se ha realizado *el diseño* del juego que enseña este dominio, haremos un rápido recorrido por las estructuras internas de la JVM que pueda servir como base para las explicaciones posteriores.

Esta descripción es muy somera. Remitimos al lector interesado a la especificación oficial de Sun (Lindholm y Yellin, 1999) para más información.

Al igual que una máquina real, la máquina virtual de Java está dividida en varias partes. Tiene, por ejemplo, una zona de memoria para albergar objetos dinámicos o una pila con los entornos de ejecución de los métodos tal y como ahora veremos. En la especificación se detalla cada una de estas partes, las instrucciones que es capaz de ejecutar y en qué cambia cada una de ellas el estado de la máquina. Es decir, lo que se define es, aparte del formato del fichero donde se guarda el código Java compilado, el conjunto de instrucciones que posee la máquina y su cometido. Y eso es lo que debe ser capaz de realizar una implementación de la JVM: leer los ficheros e implementar exactamente la semántica del código objeto que contiene, de acuerdo a la especificación.

Sin embargo, no hay restricciones a la hora de decidir cómo implementar una JVM. Una implementación podría coger la especificación literalmente, y para cada estructura de datos vislumbrada, construir un tipo de datos o una clase que la represente. También podría traducir, en tiempo de carga de cada clase, el código de cada instrucción de la JVM en instrucciones nativas de la CPU donde se va a ejecutar. Una última alternativa bastante habitual es la conocida como compilación JIT (*Just In Time*, en el momento), que realiza un análisis *en ejecución* de las porciones de código JVM que más se ejecutan para sólo traducir a código nativo las partes que más ahorro en tiempo supondrán.

5.3.1. Tipos de datos

Igual que en el lenguaje Java, en la máquina virtual existen dos grandes categorías de tipos de datos: los tipos primitivos y las *referencias*. Los tipos primitivos son los mismos que los disponibles en Java, y tienen los mismos rangos. Es decir, se dispone de los tipos `boolean` (con valores posibles `true` o `false`), `byte`, `short`, `int`, `long` (enteros con signo en complemento a 2 de 8, 16, 32 y 64 bits respectivamente), `char` (entero sin signo de 16 bits que representa un carácter en codificación UNICODE) y `float` y `double`, que son números reales de 32 y 64 bits IEEE 754.

No obstante, aunque el tipo `boolean` existe, se da poco soporte para él: no existe ninguna instrucción en la JVM que trate con este tipo de valores. Las expresiones son compiladas utilizando instrucciones asociadas al tipo entero, donde el valor 0 representa `false`.

Aunque el lenguaje Java es fuertemente tipado, no ocurre lo mismo con la máquina virtual. Ésta espera que casi todas las comprobaciones de tipos se hagan *antes* de lanzar la ejecución, por el compilador primero y por la propia máquina virtual en *tiempo de carga* de las clases después. Por eso, en las implementaciones los valores de tipos primitivos no están marcados con

el tipo que tienen. Cuando la máquina ejecuta una instrucción que suma dos enteros, coge los valores de la memoria y hace la suma, sobreentendiendo que esos dos valores referenciados en los operandos eran, efectivamente, enteros.

Existe otro tipo primitivo que no tiene correspondencia con ningún tipo en el lenguaje, pero que es utilizado en tres instrucciones de salto, el llamado `returnAddress` que representa un puntero al *opcode* (código de operación) de una instrucción. Como este tipo no tiene visibilidad desde el código fuente, sus valores *no* pueden cambiarse en ejecución, y son utilizados por el compilador únicamente para mantener lugares de salto o de retorno.

Las referencias, igual que en el lenguaje, pueden serlo a objetos, arrays o interfaces. Sus valores son referencias a elementos creados dinámicamente o a `null`. Existe distinción entre referencias a objetos y a arrays debido a que, si bien los arrays también son objetos, la máquina virtual posee instrucciones específicas para crearlos y acceder a ellos. Por ejemplo, el campo `length` de los arrays para obtener su tamaño, no es accesible a través de la instrucción habitual de la JVM utilizada para acceder a los campos de los objetos (`getfield`), sino una creada específicamente para este cometido (`arraylength`).

En cualquier caso, las referencias pueden verse como punteros a las estructuras de datos que guardan información de los objetos. Aunque la JVM no necesita guardar para cada referencia de qué tipo es, si necesita saber a qué clase pertenece el objeto al que apunta.

A modo de curiosidad, desde el punto de vista de la especificación (tanto del lenguaje Java como de la JVM) `null` *es un tipo* por sí mismo, *no un valor* válido para las variables de tipo referencia. El *tipo* `null` no es considerado ni tipo primitivo ni tipo referencia, y el único valor que puede tomar es el valor nulo. No obstante, puede ser "convertido" a cualquiera de los tipos referencia anteriores (objeto, interfaz y array) y cuando ocurre eso, se sobreentiende que la referencia en cuestión "no apunta a ningún sitio", es decir, no tiene ningún valor asociado.

5.3.2. Áreas de datos en tiempo de ejecución

Las instrucciones de la JVM actúan sobre diferentes partes de la estructura interna de la propia máquina. Para poder comprender esas instrucciones es por tanto necesario entender las diferentes partes que componen la JVM. Algunas de ellas son específicas de cada hebra, mientras que otras son comunes a todas las existentes. Por tanto, algunas de las estructuras se crean al principio de la ejecución de la máquina virtual, y otras se construyen bajo demanda, cuando la aplicación crea nuevas hebras:

- *Heap*: es una zona global, común a todas las hebras, donde se guardan las instancias de las clases (objetos) y los arrays creados. En la especificación no se establece ninguna restricción sobre cómo manejar este

espacio de memoria (si es de tamaño fijo o no, etc.), así como no se hace mención a cómo almacenar cada una de las instancias de las clases. Sí indica, sin embargo, que debe ser controlado por un recolector de basura, que elimine los objetos que ya no sean accesibles desde la aplicación.

- *Área de métodos*: es donde se guarda el código de los métodos de cada una de las clases cargadas en la máquina virtual.
- *Pilas de la JVM*: cada una de las hebras tienen una pila. En ella se guardan *frames*, que veremos más adelante. Cada pila es privada a una hebra.
- *Contador de programa*: también es privado para cada hebra, e indica qué instrucción se está ejecutando. En cada método se reinicia la cuenta, es decir, la primera instrucción de un método es siempre la instrucción `0`. El contador de programa indica qué instrucción se debe ejecutar dentro del método actual. La información sobre cuál es el método no se guarda en el contador de programa, sino en cada *frame*.
- *Área de constantes*: en cada fichero `class` existe un área de constantes con nombres y valores referenciados en los campos de dichas clases o desde las instrucciones de cada método. De esta forma, una instrucción de llamada a un método, hace referencia a una entrada en esa tabla de constantes que contendrá el nombre del método. Cada una de esas tablas se guarda en esta zona.
- *Pila para métodos nativos*: la máquina virtual permite hacer llamadas a métodos no implementados en Java, cargados de una librería externa. Estos métodos nativos necesitaran su propia pila de ejecución, por tanto cada hebra posee una de ellas.

Los *frames* son el elemento más importante para la ejecución de los métodos de Java. Son similares en esencia a los registros de activación, permitiendo almacenar datos, resultados parciales, valores devueltos y lanzar excepciones.

Cada vez que se llama a un método, un nuevo *frame* es creado en la pila de la hebra. Ese *frame* posee un área para almacenar las variables locales del método, una pila de operandos y una referencia a la tabla de constantes del método que se está ejecutando.

El número de las variables locales y el tamaño máximo usado en la pila de operandos se conoce en tiempo de compilación, por lo que es almacenado en el fichero `.class`. En tiempo de carga, no obstante, estos valores son analizados en una fase de "comprobación de sanidad" (*sanity check*) de los ficheros; una vez validados, se utilizan para hacer el menor uso de memoria posible.

Una vez más, debemos hacer notar que estas estructuras son las que marca la especificación. Dicho documento usa estas estructuras como eje principal de sus explicaciones, detallando con qué valores son inicializadas, cuándo se escribe en ellas y cuando se utilizan. Sin embargo, la implementación particular de una JVM puede prescindir de alguna de las estructuras debido a que la funcionalidad que dan ha sido programada de otra forma. Por ejemplo, en muchas implementaciones el área de constantes es suprimida; los campos de las instrucciones, en vez de tener una referencia a ese área, contienen directamente la constante, para un acceso más rápido. Si el funcionamiento de cara al exterior es el ordenado por la especificación, se puede decir que la implementación la cumple, independientemente de si internamente tiene las estructuras descritas o no.

5.3.3. Conjunto de instrucciones

En lo referente a las instrucciones aritmético-lógicas, la JVM es una *máquina de pila*. La pila utilizada para obtener y depositar los operandos se encuentra en el *frame* (registro de activación) actual. Así, por ejemplo, la instrucción de suma obtiene sus dos operandos de dicha pila, y deposita también en ella el resultado.

En realidad, debido a la gran cantidad de tipos de datos nativos, muchas instrucciones se encuentran repetidas para cubrir todos los tipos posibles. Así, se dispone de una instrucción de suma sobre `int`, sobre `long`, sobre `float` y sobre `double`. Lo mismo ocurre con las demás operaciones aritméticas, y con las lógicas y de desplazamiento, aunque esta vez sólo sobre dos de los tipos enteros (`int` y `long`).

Como se ha dicho, las variables locales se mantienen también en el *frame* al que pertenecen. Existen instrucciones para transferir valores desde ellas a la pila de operandos actual, que también se encuentran repetidas para cubrir los diferentes tipos de datos. En este caso, a los cuatro tipos numéricos anteriores se le añade el tipo referencia. En principio, esas instrucciones de carga y almacenamiento de variables locales tienen un operando para indicar el índice de la variable local en juego. Sin embargo, existen, de nuevo, instrucciones específicas para índices habituales, consiguiendo así instrucciones sin operandos. Por ejemplo, la instrucción `iload` recibe un parámetro con el índice de una variable local del *frame* actual de tipo entero cuyo valor se añadirá a la pila. Pero también existen las instrucciones `iload_0`, `iload_1`, `iload_2` y `iload_3`, en las que el valor de dicho operando está implícito en la propia instrucción.

También hay instrucciones para apilar valores constantes. Y, como antes, existe la instrucción genérica con un operando (por ejemplo `bipush`) o con el valor implícito (`iconst_0`, `iconst_1`, etcétera).

Como no podía ser de otra manera, existen instrucciones para convertir

valores de un tipo a otro, hasta un total de 15 diferentes. También hay varias para gestionar la pila de operandos, duplicando o eliminando elementos, e instrucciones de salto, tanto condicionales como incondicionales.

Por último, existe un grupo de instrucciones para acceso a arrays y objetos, que son elementos de primer orden en la JVM (en lugar de tener que ser manejados mediante estructuras de memoria manuales desde el punto de vista del código objeto). Así, hay instrucciones para crear objetos y arrays (aunque no para destruir), para acceder a sus atributos o elementos (repetidas para cada uno de los tipos posibles) o para invocar a métodos (en cuyo caso la propia JVM se encargará de la vinculación dinámica). Por último, existe una instrucción para la generación de excepciones.

5.4. JV^{2}M: la metáfora

Para poder enseñar a través de ejercicios el dominio descrito en la sección 5.2 no es en realidad necesario envolver la aplicación dentro de un juego. Bastaría un programa que mostrara el código Java de un ejercicio y diera la posibilidad al usuario de escribir (o construir mediante arrastrar y soltar) el código objeto asociado a él.

Sin embargo, *no* podemos asumir que el estudiante *conoce* el funcionamiento interno de la máquina virtual descrito en la sección anterior. Aprovechando este hecho, convertimos nuestro sistema en un *simulador*, de modo que el estudiante en lugar de limitarse a escribir las instrucciones, *las ejecuta* sobre él.

Este simulador *manipulable por el usuario* podríamos realizarlo con un interfaz WIMP tradicional, plagado de controles, barras de herramientas y el tipo de interacción que uno espera en este tipo de aplicaciones. Por desgracia, el problema de una aplicación de este estilo está en la escasez de motivación que genera; al alumno sólo le resultará llamativo su uso si está lo suficientemente interesado desde el principio. Para mejorar esta situación, nuestra propuesta es, naturalmente, envolver la labor educativa dentro de un videojuego, difuminando las labores propias del dominio que se enseña con los ingredientes del juego que añadan una pizca de diversión.

En concreto, si queremos acercarnos a las mecánicas habituales en este tipo de software de entretenimiento, debemos añadir un *entorno virtual* y un *avatar* manejado por el jugador con el que se identifique dentro de dicho entorno. Como, al fin y al cabo, lo que queremos seguir haciendo es simular la JVM y que el usuario *la manipule*, tenemos que diseñar un entorno virtual en el que realizar una asociación entre los diferentes elementos y las estructuras de una JVM típica. De esta forma, las acciones que el estudiante realice sobre los objetos virtuales ocasionarán cambios inmediatos en la JVM subyacente que se está simulando. Debido a esto, decimos que el entorno de juego representa una JVM *metafórica*. En realidad, la metáfora que hemos

construido deja fuera muchas funcionalidades de la máquina virtual, debido a que no pretendemos que con nuestro sistema se puedan aprender todos sus detalles. Por ejemplo, no se simulan las hebras, las clases son cargadas por completo al principio de cada ejercicio y no soportamos excepciones.

Al abandonar los familiares interfaces WIMP y adentrarnos en entornos más complejos podemos sufrir el problema de incrementar el tiempo para comprender su uso. Esta dificultad añadida no puede despreciarse. De hecho, McKenna y Laycock (2004) creen que los sistemas educativos que siguen una aproximación *conductista* son mucho más sencillos de utilizar porque las interacciones son más simples y los usuarios están más acostumbrados a ellas. Al desplazarnos hacia el constructivismo, la necesidad de una interacción más sofisticada es una traba para los usuarios menos avezados, pues los entornos inmersivos demandan más tiempo y un esfuerzo cognitivo mayor.

En nuestro caso, este problema lo suplimos utilizando modos de interacción *habituales en los videojuegos*, con la esperanza de que los usuarios los conozcan de antemano y se sientan cómodos con ellos. Esto suaviza la curva de aprendizaje para las operaciones básicas, como desplazarse por el entorno o manipularlo. Así, la parte *conductista* que Siang y Rao (2003) identificaron como primera fase del aprendizaje de las reglas de un juego (mencionado en la página 96) desaparece. De esta forma no existirá un escalón inicial para acostumbrarse al interfaz, al igual que no lo hay en el caso de usar un interfaz WIMP.

En concreto, el modelo de interacción que nosotros hemos escogido es similar al del género de *juegos de aventura* descritos brevemente en la sección 3.3.1[1]. En este género destaca con especial fuerza LucasArts Entertainment Company LLC, creadora de SCUMM (*Script Creation Utility for Maniac Mansion*, Utilidad de Creación de Guiones para Maniac Mansion) y GrimE (*Grim Edit*), dos de los motores para el desarrollo de juegos de aventuras más conocidos (figura 5.1). Ambos se basan en *personajes* y *objetos* dinámicos dibujados sobre un fondo estático, y que interactúan entre sí por medio de una serie de primitivas básicas. La diferencia principal entre ambos estriba en que el segundo se basa completamente en tecnología 3D.

En ambos casos, el jugador controla a uno de los personajes del entorno, y dispone de un *inventario* donde conserva aquellos objetos que haya recogido a lo largo del tiempo. Es posible aplicar sobre los elementos dinámicos del entorno y sobre los objetos del inventario las *acciones primitivas* proporcionadas por el sistema. Éstas fueron disminuyendo y simplificándose con el tiempo, de modo que finalmente sólo han sobrevivido cuatro acciones, que son las que nosotros hemos incluido en nuestro proyecto. Cuando el personaje se sitúa en las cercanías de un objeto, se le da la oportunidad de realizar

[1] Esta elección es coherente con el resultado de Amory et al. (1999) que mencionamos en la sección 3.8.2. Concluyeron que eran, precisamente, los juegos de *aventura* los que preferían sus alumnos.

(a) Mokey Island 1 (SCUMM)

(b) Mokey Island 4 (GrimE)

Figura 5.1: Monkey Island: de SCUMM a GrimE

sobre él alguna de las acciones. El significado concreto dependerá del objeto en cuestión aunque, de manera general, las opciones son:

- *Mirar*: el propio avatar del jugador proporciona una descripción corta del objeto sobre el que se realiza esta acción. Esto proporciona un primer nivel de ayuda sobre el entorno.
- *Coger*: el avatar recoge, si es posible, el objeto seleccionado y lo mantiene en su inventario para un uso posterior.
- *Usar*: el objeto es utilizado de un modo particular, dependiendo del tipo de objeto.
- *Usar con*: permite *combinar* el objeto seleccionado actualmente en el inventario con el objeto activo del entorno sobre el que se realiza la acción.

Como ya se ha comentado, todo el funcionamiento del entorno se sustenta en un *simulador* de la JVM, y en una serie de relaciones implícitas en el entorno que relacionan cada elemento virtual con uno de la JVM. Para nuestro propósito, la máquina virtual de Java tiene un área que almacena todas las clases e interfaces cargados, donde se almacenan las relaciones entre ellos, sus métodos y los atributos estáticos. Posee además un *heap* o montón donde se guardan todos los objetos creados en la ejecución del programa, y una pila de *frames*, que guardan los contextos de ejecución de cada método, lo que incluye tanto las variables locales como la pila de operandos con resultados parciales de operaciones.

Nuestra metáfora es básicamente un escenario exterior, con múltiples edificios, que simbolizan cada una de las clases cargadas, objetos creados y frames almacenados en la JVM. De hecho, a grandes rasgos, el entorno está separado en tres partes: el *barrio de clases*, el *barrio de objetos*, y el *frame*. En los dos barrios hay edificios, a los que el usuario podrá entrar

para ver y manipular los elementos que contienen. También el *frame* es una construcción, pero con un interior particularmente importante en la ejecución de programas, por lo que se mantiene separado y fuera de los otros dos barrios.

En el barrio de las clases se encuentra un edificio por cada clase Java cargada en la JVM subyacente. Para ahorrar detalles al estudiante, todas las clases necesarias para un ejercicio son cargadas (y mostradas) desde el principio, por lo que el estudiante no tendrá que preocuparse sobre la carga dinámica de clases que una JVM real podría realizar.

A través de los objetos virtuales dentro de cada edificio, el estudiante puede obtener el código fuente de los métodos de la clase a la que representa el edificio en cuestión, y acceder a sus atributos estáticos.

Algo parecido ocurre en el barrio de los objetos, que representa al *heap* con la memoria dinámica. Cada edificio representa un objeto, y en su interior el avatar puede conseguir los valores de los atributos (de instancia) de dicho objeto.

La entrada al barrio de los objetos está protegida por un personaje animado a quien se puede solicitar la creación de nuevos objetos así como la clase a la que pertenece un objeto ya creado.

Por último, el entorno dispone del edificio del *frame* actual, que contiene las variables locales y la pila de operandos del "registro de activación" activo. En él se deberán ejecutar la mayor parte de las instrucciones compiladas, dado que es mucho más habitual realizar operaciones aritméticas y lógicas que invocar a métodos o crear objetos.

No obstante, la máquina virtual posee en realidad una *pila de frames* (a la que antes llamábamos pila de la hebra) para mantener la información de los métodos que han quedado suspendidos a la espera de que la última invocación que han realizado a otro método finalice. Para representar esa pila en el entorno, el edificio del *frame* se "hunde" cuando se realiza una invocación, y cae un nuevo edificio similar, pero preparado para el nuevo método. Cuando la ejecución de un método termina, el edificio que representa su *frame* se eleva y resurge del suelo el anterior.

La labor principal del usuario es *traducir mentalmente* (al vuelo) el código fuente suministrado, y ejecutar las instrucciones de la JVM deducidas dentro de la máquina virtual que, metafóricamente, representa el entorno virtual. La ejecución de esas instrucciones suele requerir el intercambio y movimiento de *operandos*. Por ejemplo, la instrucción `iload_0` ya comentada supone entrar en el edificio del *frame*, conseguir el valor de la variable local de índice `0`, y colocarlo sobre la pila de operandos. El "almacén" de las variables locales y la pila de operandos son objetos del entorno que se manipulan a través de las primitivas básicas ya mencionadas. Así, por ejemplo, al ir a ejecutar el `iload_0`, el usuario tendrá en su inventario el operando implícito de la instrucción (el `0`) que *usa con* el almacén de variables locales para obtener

Figura 5.2: El usuario a punto de apilar un valor en la pila de operandos

el valor. Éste, representado mediante una caja en el entorno, es de nuevo *usado con* la pila de operandos, lo que supone que la caja (el operando) queda *apilado* (figura 5.2). En lo que se refiere al entorno las repercusiones de las acciones resultan razonablemente claras, y por debajo están causando los cambios oportunos sobre la JVM simulada.

En el apéndice A se describe con detalle los pasos que el usuario debe realizar sobre el entorno para resolver un ejercicio de ejemplo. Puede verse que ejecutar una sola instrucción sobre el entorno puede terminar resultando bastante laborioso. Esto no es un problema las primeras veces que debe ejecutarse cada instrucción, pues se desea que se ponga en práctica completamente para comprender lo que hace. Al fin y al cabo, nuestro sistema enseña tanto, de manera intuitiva, la traducción de código java a código objeto (conocimiento de "alto nivel") como la estructura y funcionamiento de la JVM (conocimiento de "bajo nivel").

El problema, es que cuando los ejercicios de compilación se van haciendo cada vez más y más complicados, forzar al usuario a continuar ejecutando las instrucciones de la JVM con todo lujo de detalles termina siendo una lacra. El alumno *ya conoce* como funcionan esas instrucciones, y en los ejercicios complicados no es necesario seguir comprobándolo, porque, de hecho, termina siendo muy pesado para el alumno. Para solucionarlo, con el avance de la dificultad de los ejercicios de compilación, algunas instrucciones de la JVM dejarán de tener que ejecutarse manualmente. En lugar de eso, van surgiendo ayudantes (en concreto "campesinos") que se encargan de los detalles que el usuario ya ha demostrado conocer. Todo esto se encuentra justificado por la *historia de fondo* del programa, que da sentido a la labor del estudiante. El apéndice B describe dicha historia.

5.5. JV^{2}M : el sistema a vista de pájaro

Una vez descrito la labor principal del estudiante (ejecutar *al vuelo* un código Java sobre la JVM) analizaremos de manera somera el funcionamiento del programa en relación con el esquema del capítulo 4. Naturalmente, el sistema se basa en la *resolución de ejercicios* y mantenemos la fusión entre un ejercicio y un nivel del juego, algo que queda implícito en la descripción del mencionado apéndice B.

La primera de las etapas del esquema era la elección de los conceptos a practicar, en la que el sistema de tutoría decide, en función de los conocimientos atribuidos al usuario, qué conceptos del dominio que se enseña se deben poner en práctica en la siguiente iteración.

Para eso, nuestro sistema dispone de información sobre esos conceptos, de la que hablaremos en la sección 5.6. Es necesario mantener un orden de dificultad en todos ellos para que el sistema conozca la mejor secuencia para enseñarlos. El modelo del estudiante se basa en una sencilla plantilla, que sigue la pista de qué conceptos conoce ya y cuales aún no se han descrito, de forma que resulta sencillo decidir qué se debe practicar en el siguiente ejercicio.

En este caso, por lo tanto, el conocimiento del que dispone el sistema no es demasiado sofisticado. De hecho, no hay como tal *conocimiento pedagógico* que se utilice para realizar una toma de decisiones informada en función de los diferentes estilos de aprendizaje. En lugar de eso, el sistema sencillamente mantiene *conocimiento cableado* sobre la mejor secuencia, y la sigue ciegamente sin tener la posibilidad de replantearse si es realmente la mejor opción para el estudiante actual.

Una vez escogidos los conceptos a practicar, es necesario obtener un ejercicio que los ponga en práctica. Es importante que el ejercicio propuesto *no* incluya conceptos más complicados que no se hayan pedido explícitamente para evitar que el alumno tenga que enfrentarse a una complejidad excesiva.

Comentábamos en la sección 4.5 que en principio aquí hay dos posibilidades: disponer de una base de ejercicios o de un sistema experto que genere los ejercicios al vuelo.

En este caso, nos hemos decantado por la opción de disponer de una batería de ejercicios escritos por un experto en el dominio. Los ejercicios se encuentran enlazados de acuerdo a los conceptos que ponen en práctica. La selección consiste, por lo tanto, en una búsqueda sobre la base de casos en función de los conceptos solicitados por la etapa anterior.

La resolución del ejercicio por parte del estudiante se realiza en el entorno virtual proporcionado por el sistema. Como ya hemos descrito, esta resolución se encuentra diluida con mecánicas tradicionales del mundo de los videojuegos para dar un cierto carácter lúdico a la labor de aprender.

En el capítulo 4 planteábamos también varias alternativas para detectar si la solución suministrada por el usuario era o no correcta. El primer punto importante a decidir es si el sistema esperará hasta el final del episodio de resolución para analizar el resultado, o irá realizando una comprobación incremental que le permita tomar medidas *durante* la resolución.

En nuestro caso, la comprobación se realiza *continuamente*, por lo que el sistema monitoriza al estudiante durante todo el proceso de resolución para controlar si está o no dando los pasos adecuados.

Por otro lado, también comentábamos que la solución del ejercicio podía conseguirse desde diferentes fuentes. En concreto, puede *crearse* dinámicamente a partir del enunciado propuesto al estudiante, o bien, en el caso de que el ejercicio se extraiga de una base de casos, conseguirse *a la vez* que el propio enunciado. Aquí nos hemos decantado por esta última opción, de manera que cuando se extrae de la base de ejercicios el siguiente que se practicará a partir de los conceptos escogidos en la primera fase, éste trae consigo automáticamente su solución. De esta forma, la base de ejercicios está indexada por conceptos, y contiene tanto el ejercicio como su solución.

En realidad esta decisión puede resultar un tanto chocante. En el capítulo anterior comentábamos que normalmente los ejercicios y sus soluciones se emparejaban de manera manual cuando resultaba muy complicado construir un resolutor automático de los ejercicios. Al fin y al cabo, exigir que los expertos humanos planteen ejercicios *y los resuelvan* supone un incremento en el trabajo manual bastante significativo.

En nuestro dominio, podríamos ahorrarnos este trabajo porque, después de todo, *disponemos* de un sistema experto que es capaz de resolvernos cualquier ejercicio que podamos plantear al estudiante: el compilador de Java `javac`. Al fin y al cabo, lo que estamos enseñando es, de manera intuitiva, cuál es el resultado de dicho compilador para cada posible programa fuente, por lo que ¿qué mejor modo de comprobar una solución que comparándola con la proporcionada por el propio compilador?.

El problema es que `javac` *no* es un *sistema experto transparente* que se pueda aprovechar para obtener conclusiones sobre *las causas de los errores* que el estudiante pueda cometer. Es decir, *siempre nos da una solución correcta*, pero no hay modo de bucear en su funcionamiento interno para indagar sobre las desviaciones en la compilación que los usuarios podrían realizar. Además, no hay forma de saber por qué una compilación es correcta para explicársela al estudiante.

Debido a ello, la solución se almacena de manera conjunta con cada ejercicio. Aunque la solución proporcionada está basada en el resultado generado por `javac`, en general contiene información adicional que resulta útil en el proceso de tutoría. En la sección 5.7 se proporciona más información sobre todo esto.

Naturalmente, un sistema educativo no está completo si no es capaz

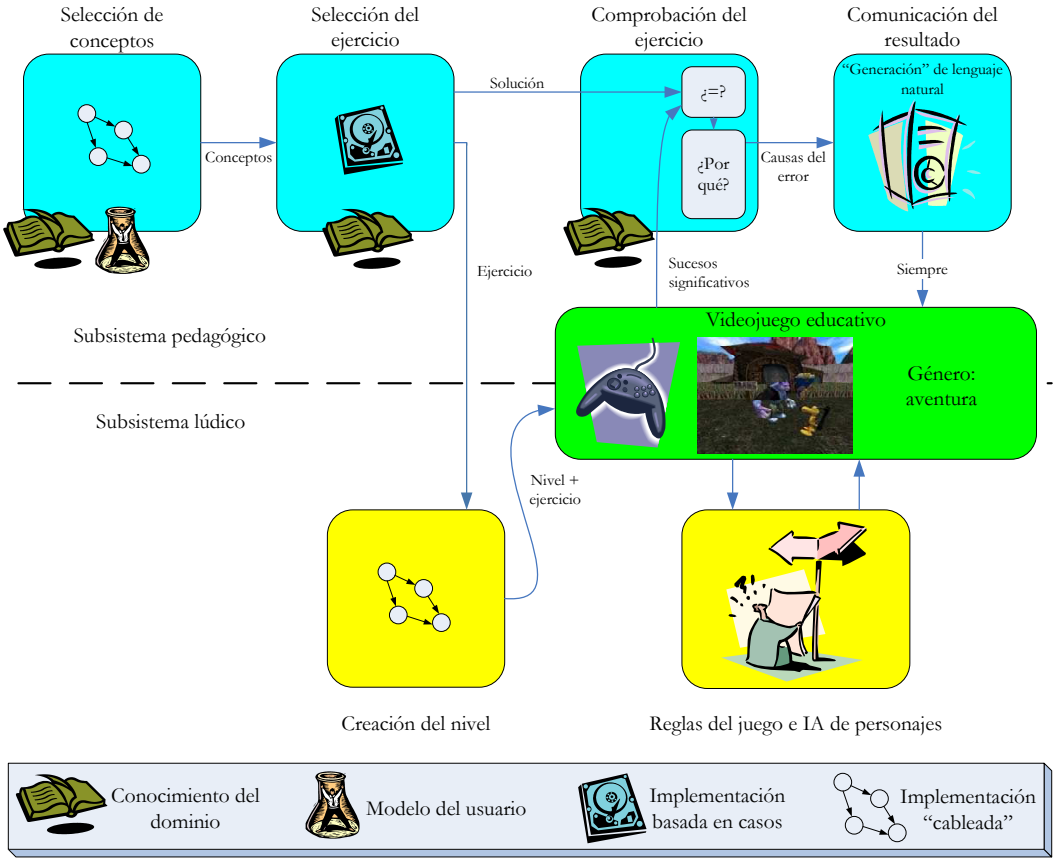


Figura 5.3: Esquema lógico del funcionamiento de JV²M

de suministrar información sobre por qué algo está bien o mal. En nuestro caso utilizamos un agente pedagógico llamado JAVY, cuya existencia queda justificada en la historia de fondo descrita en el apéndice B. Dado que la comprobación de la solución se realiza continuamente, el agente es capaz de proporcionar información *contextualizada* en cualquier momento, adaptada a la situación concreta en la que se esté. Por simplicidad, *no* permitimos que el estudiante se desvíe de la solución correcta, por lo que el agente pedagógico es *proactivo*. Cuando detecta algún problema en la resolución, se anticipa y da pistas y ayuda no solicitada para evitar que el alumno pierda tiempo siguiendo un camino incorrecto. Esta *realimentación inmediata* suministrada por el sistema es, en el fondo, comparable a la que daba la máquina de Pressey, y que tanto agradaba a Skinner.

La figura 5.3 muestra el esquema con el funcionamiento a alto nivel de la aplicación. Es una particularización del modelo descrito en el capítulo anterior, y que quedó también resumido en la figura 4.5. Vemos que, efec-

tivamente, la primera etapa se realiza sin utilizar ninguna técnica de Inteligencia Artificial, que los ejercicios se extraen de una base de casos junto con su solución, y que la comunicación con el estudiante se realiza *siempre*, sin necesidad de que el usuario pida ayuda explícitamente. Respecto a la parte de videojuego, el sistema *no* ajusta la dificultad del nivel en lo referente a sus aspectos meramente lúdicos. Por otra parte, dicho nivel es construido de manera procedimental a partir del ejercicio, principalmente inicializando el *barrio de clases* con aquellos "edificios" que representan a las clases Java contenidas en él.

Por completitud, antes de entrar en los detalles de la representación del conocimiento utilizada dentro de la aplicación, merece la pena detenerse y categorizar nuestro videojuego educativo en función de los diferentes *ejes* que describimos en la sección 3.8.

El objetivo último del programa es *enseñar* (y no meramente *repasar*) el proceso de traducción, con un mínimo apoyo externo por parte de un profesor[2]. El dominio enseñado es *lineal* dado que es posible colocar sus diferentes conceptos en orden (parcial) de dificultad. De hecho, la elección del siguiente episodio de aprendizaje hace uso de esto para ir introduciendo de manera paulatina aspectos cada vez más complejos del proceso de traducción.

Naturalmente, se trata de un proyecto *de investigación* que, no obstante, no olvida a sus usuarios objetivo. Decíamos en su momento que este tipo de proyectos tienen una dificultad de desarrollo considerable debido al arduo esfuerzo de desarrollo inicial necesario para conseguir empezar a ver algún tipo de resultado. En este marco, una opción razonable sería aprovecharse de algún tipo de *motor* para realizar *juegos de aventura* que estuviera disponible en el mercado (desarrollando así, un *mod*). Por desgracia, la necesidad de creación del nivel *al vuelo* (a partir del ejercicio elegido), los requisitos relativamente sofisticados de toma de decisiones por parte de los diferentes personajes, o la necesidad de *simular* por debajo una JVM impiden utilizar las relativamente sencillas opciones. El resultado es que el programa ha sido desarrollado por completo a lo largo de varios años dentro del grupo de investigación en el que se encuadra este trabajo, haciendo uso, eso sí, de motores de juego genéricos.

5.6. La jerarquía conceptual

Pero recuperemos de nuevo las cinco fases del ciclo de ejecución tal y como se han planteado al principio de la sección anterior. Para conseguir llevarlas a cabo del modo descrito, el sistema necesita tres grandes bloques de conocimiento sobre el dominio, que se utilizan simultáneamente en todas

[2]Esto se realiza, naturalmente, asumiendo la precondición de que el usuario conoce el lenguaje Java.

las etapas (Gómez Martín et al., 2003, 2005d, 2007a).

El primer nivel de conocimiento consiste en una enumeración de aquellos conceptos del dominio que el usuario debe aprender. Algunos sistemas (Stottler et al., 2001b) almacenan esa información utilizando un árbol. Nosotros, sin embargo, organizaremos todos nuestros conceptos en un grafo dirigido (acíclico), donde los nodos representan los conceptos y las aristas las relaciones entre ellos. A esta estructura interrelacionada de conceptos la denominamos *jerarquía conceptual*.

Los conceptos mantienen información genérica sobre su significado. Son utilizados en dos lugares del sistema:

- *Modelo del usuario*: debido a que los conceptos son precisamente el material que se espera que el alumno aprenda, un modo natural de almacenar su conocimiento es utilizar el modelo de estudiante basado en plantillas marcando cuales de los conceptos se suponen ya conocidos y cuales no. La información almacenada acerca de cada uno de los conceptos que se deben aprender puede ser algo más elaborado que un simple "ya lo sabe" o "aún no lo sabe". Es posible almacenar, por ejemplo, información sobre el porcentaje de veces que ha aplicado correctamente el concepto en los últimos ejercicios en los que era necesario. Esto permite marcar la diferencia entre "aún no lo sabe porque no ha sido explicado y nunca ha tenido que usarlo" y "aún no lo sabe, porque no lo ha conseguido poner en práctica correctamente las últimas 4 veces".

- *Módulo pedagógico*: en la fase de elección de los siguientes conceptos a practicar es necesario disponer, precisamente, de la lista de conceptos posible. En JV^{2}M ese papel lo juega esta jerarquía conceptual. Además, para poder tomar la decisión sobre el siguiente paso a dar se necesita lo que algunos autores denominaban *conocimiento del currículo* (Kopec y Thompson, 1992, página 93). Así, podemos tener anotados los conceptos para marcar la mejor secuencia de enseñanza.

Los conceptos de la jerarquía son por lo tanto el medio de intercambiar información entre la primera y la segunda etapa de nuestro ciclo. Además, los ejercicios se encuentran, por tanto, *indexados* en la base de ejercicios de acuerdo precisamente a estos conceptos. Así, si en la primera etapa se elige practicar, pongamos, la compilación del `if`, se realizará una búsqueda sobre los ejercicios para obtener uno que ponga en práctica el concepto `if`.

La figura 5.4 muestra una mínima parte de la jerarquía conceptual. Los conceptos y las relaciones que aparecen son ligeramente diferentes en la jerarquía utilizada por el sistema, que ha sido simplificada aquí para no complicar innecesariamente el ejemplo. Hemos identificado cinco grandes tipos de conceptos de acuerdo a su nivel de abstracción del conocimiento que contienen:

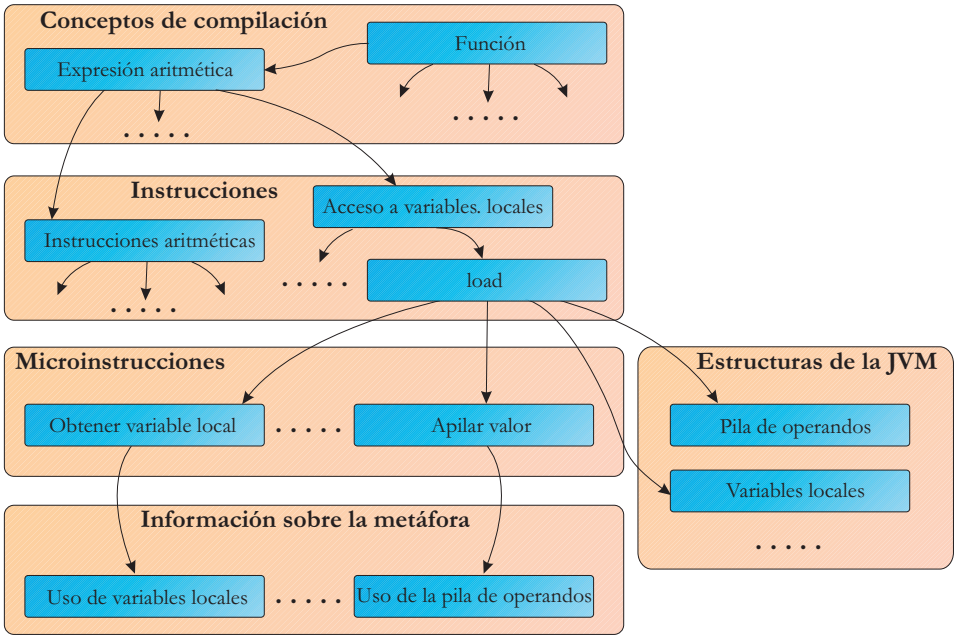


Figura 5.4: Ejemplo con la estructura de la jerarquía conceptual

- *Conceptos de compilación*: se refieren a ideas relacionadas con el objetivo último del proceso de aprendizaje, la compilación de código Java. En realidad estos conceptos son independientes del lenguaje concreto, pues muchos son aplicables de manera directa a cualquier otro lenguaje imperativo. Se refieren por lo tanto a las estructuras de alto nivel que el usuario debería aprender a compilar. Algunos ejemplos incluidos en esta colección de conceptos son la compilación de expresiones aritméticas, o la estructura `if`.

- *Instrucciones de la máquina virtual*: este grupo incluye, obviamente, las instrucciones de la JVM. La máquina virtual puede ejecutar más de 200 instrucciones. Como se ha dicho, muchas de ellas varían muy poco entre sí, por ejemplo sólo en el tipo de los operandos. Para representar esta característica, este grupo de conceptos se almacena ordenado en una jerarquía, con los conceptos referentes a las instrucciones en el nivel inferior, y relacionadas mediante conceptos superiores que las agrupan según sus tipos, como por ejemplo instrucciones aritméticas o de invocación a métodos.

- *Microinstrucciones*: dentro del entorno virtual, tanto Javy como el estudiante deben *ejecutar* las instrucciones del código en ensamblador resultado de compilar el ejercicio. La ejecución de cada instrucción no es una acción atómica, sino que a menudo requiere varios pasos más

pequeños, habitualmente conocidos en el ámbito de la arquitectura de ordenadores por el término *microinstrucciones*. Cada una de ellas suele realizar un pequeño cambio en el estado de la máquina virtual, como apilar un elemento, o modificar el valor de una variable local. En nuestro entorno metafórico, algunas de las microinstrucciones son utilizadas para obtener objetos que almacenan de alguna forma información del estado de la JVM, y que pasan a formar parte del inventario para utilizarlos en pasos posteriores. Ejemplos dentro de este grupo son las microinstrucciones para leer el valor de una variable, u obtener un método de un objeto. Este grupo de conceptos almacena por lo tanto información sobre cada una de esas operaciones primitivas que el usuario o el agente pedagógico pueden ejecutar. Cada concepto instrucción del grupo anterior está relacionado con las microinstrucciones de este grupo necesarias para ejecutarla.

- *Estructuras de la máquina virtual*: cada una de las partes de la JVM posee un concepto asociado dentro de este grupo. Ejemplos de este tipo son "pila de operandos", "pila de frames" o "heap". Las instrucciones y microinstrucciones que manejan alguna estructura de la JVM estarán relacionadas con su correspondiente concepto dentro de este grupo.

- *Operaciones del interfaz del usuario*: las microinstrucciones almacenadas en el tercer grupo guardan información sobre sus efectos sobre la máquina virtual y el entorno virtual, pero no sobre el modo en el que se llevan a cabo a través de las tres operaciones del interfaz de usuario (coger, usar y usar con)[3]. La información sobre cómo realizar una microinstrucción se encuentra almacenada en este grupo. Así, por ejemplo, existe un concepto "Usar pila de operandos" (con otro objeto), que estará relacionado con la instrucción primitiva "apilar un valor".

El modelo del usuario y el módulo pedagógico (la base de ejercicios) sólo hacen uso de los dos primeros grupos, con los conceptos sobre compilación y las instrucciones de la JVM. El resto de grupos es interesante porque la jerarquía conceptual en su conjunto es utilizada también para proporcionar explicaciones, ya sea bajo petición expresa del alumno o por iniciativa propia del sistema al descubrir alguna deficiencia. Para eso, aparte del nombre y el grupo al que pertenece (compilación, instrucción, microinstrucción, etcétera), cada concepto dispone de una *descripción textual* que será utilizada por el agente pedagógico para explicarlo. Así, si el usuario solicita ayuda sobre un concepto particular, JAVY le dará ("*leerá*") su descripción, y ofrece al usuario la posibilidad de preguntar por los conceptos relacionados con el

[3]En realidad el usuario dispone de una operación más, *mirar*, pero que no afecta al entorno virtual (es en realidad un modo de obtener ayuda instantánea), por lo que para ejecutar las microinstrucciones nunca se hará uso de ella.

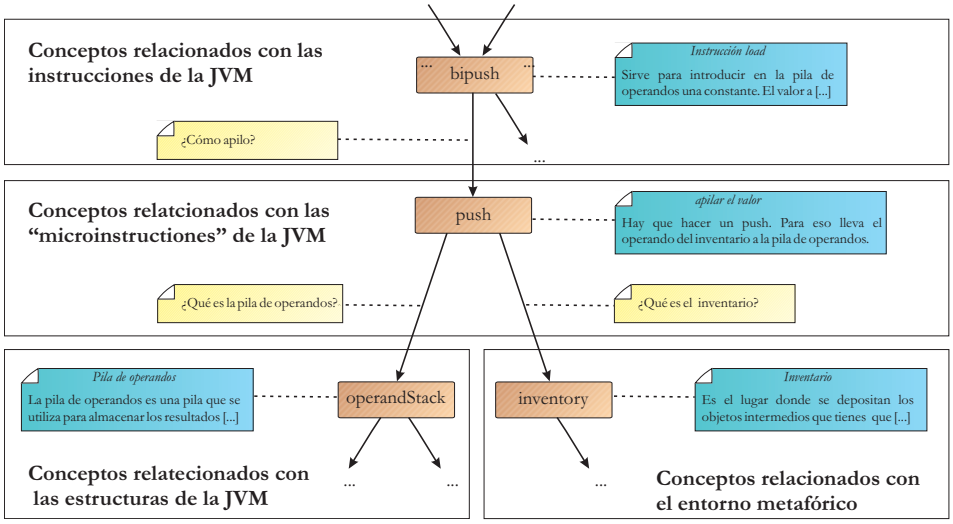


Figura 5.5: Fragmento detallado de la jerarquía conceptual

actual a través de las conexiones que aparecen en la jerarquía. Para eso, en lugar de utilizar un intérprete de lenguaje natural que analice esas preguntas, las *aristas* que relacionan conceptos en la jerarquía están etiquetadas con el propio texto de la pregunta, y el sistema da al usuario la opción de elegir cualquiera de las disponibles.

La figura 5.5 ejemplifica la idea. Por ejemplo si se le pregunta a JAVY sobre el concepto `push`, da la explicación que aparece en la etiqueta de ese concepto: "Hay que hacer un push. Para eso, lleva el operando del inventario a la pila de operandos". Después de eso, se comprueban los enlaces, y se le permite al usuario preguntar "¿Qué es la pila de operandos?" y "¿Qué es el inventario?". El ciclo de la conversación se repite utilizando otro concepto, dependiendo de la selección del usuario.

Los estudiantes encuentran bastante molesto que el agente pedagógico repita la misma explicación por duplicado (Stone y Lester, 1996), aparte de que es algo que le hace perder buena parte de su credibilidad. Para evitar esa situación, cada concepto tiene además una segunda descripción más corta que será utilizada cuando la primera haya sido presentada recientemente al usuario. Estas explicaciones secundarias serán del estilo de "Recuerda que...".

La jerarquía conceptual ha sido desarrollada con una herramienta de autoría propia (llamada JaCo) que se adapta a las particularidades de la información que guardamos por cada concepto (figura 5.6). La información se almacena en un fichero XML que es luego leído por el sistema para construir el conocimiento en su propia representación interna. En la sección C.1.1 aparece la DTD (*Document Type Definition*, Definición de tipo de documento)

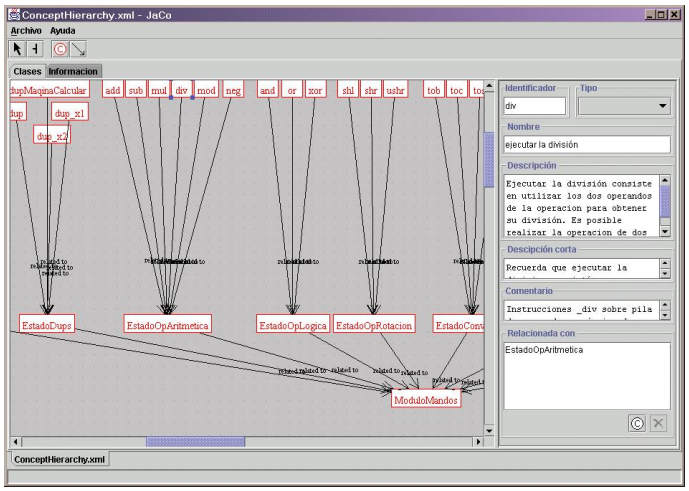


Figura 5.6: JaCo en ejecución

seguida por dicho XML. Aparte de guardar por cada concepto la información ya descrita, también permite al experto almacenar una explicación en lenguaje natural para cada uno a modo de documentación. Esta información *no* será utilizada por el sistema en ningún momento, y se desprecia en tiempo de carga; como sólo es útil durante la creación de la jerarquía conceptual, sólo está disponible desde la herramienta. En la sección C.1.2 se muestra el XML asociado a la porción de la jerarquía de la figura 5.5.

5.7. Los ejercicios

Como se ha dicho, nuestro sistema se basa en la enseñanza basada en problemas, que son extraídos de una base de ejercicios creada por expertos del dominio. La recuperación del siguiente ejercicio a resolver se realiza en función de los conceptos escogidos y adaptados a las necesidades particulares del estudiante. Los ejercicios están, por lo tanto, indexados a partir de los conceptos de la jerarquía conceptual descrita en la sección anterior.

El enunciado de un ejercicio está compuesto por una o varias clases Java que deben compilarse. El usuario en realidad *ejecuta* el código compilado, por lo que es necesario disponer de un *punto de inicio* en la ejecución. Es decir, si sólo le pidiéramos que compilara el código Java, bastaría con proporcionarle las clases correspondientes, sin ninguna restricción adicional. Sin embargo, en nuestro caso es necesario proporcionar un punto de inicio de la ejecución. Ese papel lo juega siempre, como no podría ser de otro modo, la clase que disponga del método `main`, que es el primero que se lanza en todos los programas escritos en Java.

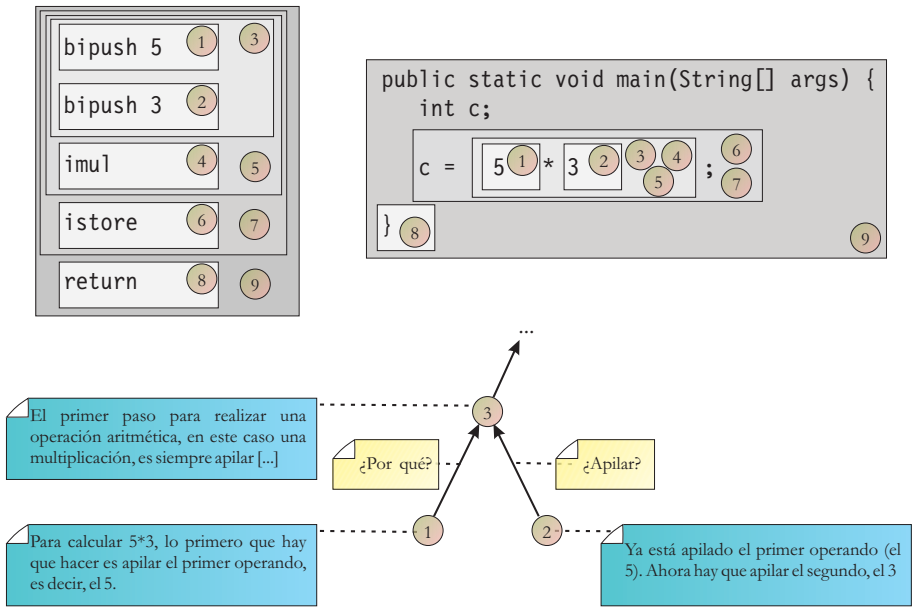


Figura 5.7: Fragmento detallado de un ejercicio

Como ya se ha comentado, junto con el código fuente que hace las veces de enunciado, también se guarda el código compilado que constituye su solución. El experto que desarrolla el ejercicio debe *anotar* manualmente la relación entre los bloques de código fuente y los de código objeto. Estas relaciones, además, no son lineales, sino jerárquicas, siguiendo una estructura de bloques de código fuente que se asemeja al árbol sintáctico del programa original. Así, por ejemplo, un bloque de código fuente relativo a una expresión de asignación estará asociado con *todas* las instrucciones de la JVM resultado de compilar dicha expresión. Pero, simultáneamente, esa expresión está dividida en diferentes partes (por ejemplo, lado izquierdo y lado derecho), cada una de ellas asociadas, de nuevo, a las instrucciones que dan lugar. Naturalmente, existirán solapamientos de código fuente y objeto entre bloques, pero eso no supone ningún problema.

Cada asociación entre un bloque de código fuente y uno de código objeto habrá sido explicado por el autor del ejercicio, indicando por qué el código fuente se ha compilado de la manera en que se ha hecho. Además, las "aristas" que unen los diferentes bloques de código en la jerarquía también se encuentran anotadas con una pregunta al igual que hacíamos con los diferentes conceptos de la jerarquía conceptual, que se utilizan para guiar la conversación.

La mejor manera de entender todo esto es a través de un ejemplo. La figura 5.7 muestra un fragmento de un ejercicio. Por claridad, hemos etiquetado

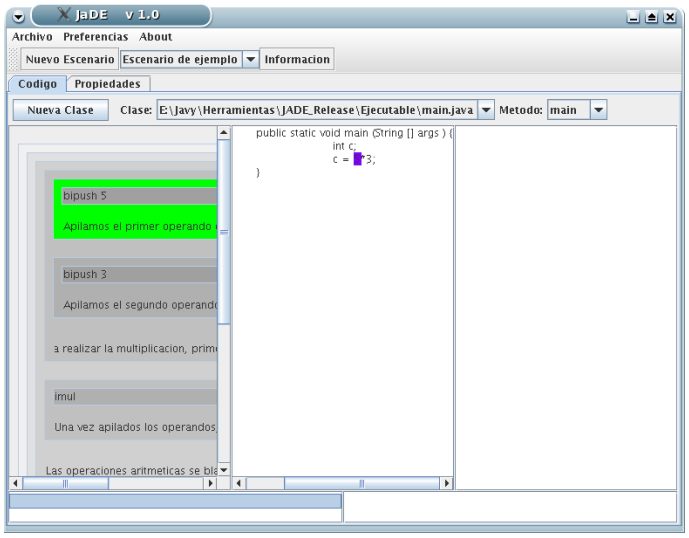


Figura 5.8: JaDe (editor de ejercicios) en ejecución

cada bloque del árbol con un número. Cuando ejecutamos la primera instrucción de la JVM (`bipush 5`), el usuario puede preguntar al agente por qué se ha traducido el primer operando de la expresión `5*3` con esa instrucción. En este caso, JAVY recuperará la explicación asociada al bloque etiquetado con un 1 en la figura, y contestará con "Para calcular 5*3, lo primero que hay que hacer es apilar el primer operando, es decir, el 5". Utilizando la arista que une dicho bloque con el número 3 del árbol, el alumno puede preguntar "¿Por qué?", lo que lo permitirá ir ascendiendo en la jerarquía y obteniendo información cada vez más general y referida a mayores porciones de código.

Crear esta estructura puede resultar bastante complejo, por lo que se ha desarrollado una herramienta (llamada JaDe) pensada específicamente para realizar esta labor (figura 5.8). El ejercicio creado es mantenido en un fichero XML que sigue la DTD de la sección C.2.1, y que será leído posteriormente por la aplicación. El ejercicio de la figura 5.7 se muestra en la sección C.2.2. La labor inicial del autor es proporcionar las clases Java, y la herramienta automáticamente invoca a `javac` para compilarlas. Esto proporciona una primera versión de la solución del ejercicio con la que comenzar a trabajar.

Dado que es posible que el compilador haya creado un programa objeto optimizado, la compilación propuesta podría no resultar obvia. El experto tiene la posibilidad de ajustar el resultado para mejorar su claridad, a costa de "estropear" las posibles eficiencias que `javac` hubiera decidido añadir. En este caso, es responsabilidad del autor del ejercicio garantizar que, efectivamente, los cambios realizados sobre el código objeto *lo mantienen correcto*; la herramienta no tiene modo de garantizar la corrección del resultado.

También en un intento de facilitar la creación de las relaciones entre código fuente y código objeto, la compilación con `javac` se realiza *en modo depuración*. Esto, por un lado, evita que se utilicen muchas de las optimizaciones mencionadas antes, lo que ahorra trabajo del especialista. Por otro, también proporciona información sobre qué instrucciones de código objeto se corresponden con cada linea de código fuente. Esta información es añadida por el compilador para que el *depurador* sepa en qué instrucciones de la JVM detenerse durante una ejecución paso a paso. El uso que hace la herramienta de esta información es totalmente diferente. En concreto, aprovecha estos datos para crear un primer grupo de *bloques* relacionando código fuente y código objeto. En este caso, *no* habrá solapamientos entre bloques, y, en conjunto, la estructura formada por todas estas relaciones será "plana". Pero sirve como un buen punto de inicio para que el tutor pueda continuar afinándola. En concreto, tendrá que crear la jerarquía de bloques completa según le convenga y anotar todas las relaciones con explicaciones. No obstante, en métodos relativamente largos esta primera información y disección del código objeto es clarificadora y ahorra tiempo al acotar las necesidades de análisis sobre el trabajo que el compilador ha hecho por nosotros[4].

Aparte de todo este trabajo sobre el código fuente, código objeto y sus relaciones, la herramienta también se utiliza para especificar los conceptos que el ejercicio pone en práctica. Para eso, durante la inicialización, carga el fichero XML con la jerarquía conceptual, averigua las instrucciones de la JVM que se utilizan en el ejercicio, y lo enlaza con sus conceptos. El creador del ejercicio tendrá que añadir a esa lista de conceptos los asociados a las estructuras del lenguaje Java que aparecen en el código fuente. De esa forma, el enlazado entre un ejercicio y los conceptos que pone en práctica se realiza de manera semiautomática.

5.8. La resolución del problema

Ya se ha comentado que el alumno en realidad *no* escribe el código objeto, sino que *lo ejecuta* dentro de la máquina virtual metafórica que representa el entorno. La ejecución de cada instrucción de la JVM (como `bipush` o `invokespecial`) no es atómica dentro del juego, sino que requiere una serie de pasos en un determinado orden. Cada acción supone un pequeño cambio dentro de la JVM subyacente, al modificar el estado de las diferentes estructuras internas de la máquina representadas en el entorno. Como ya se esbozó en la sección 5.6, a cada uno de esos pasos le hemos denominado *microinstrucción*.

En realidad, este concepto de "microinstrucción" *no* existe en la especifi-

[4]Como ejemplo, puede evaluarse el laborioso trabajo necesario para realizar esta primera separación de manera manual viendo la versión en Java y compilada del conocido algoritmo de Euclides, que se muestra en la página 231.

```
bool JVM::executeInstruction(
                    const JVMInstruction &instruction) {
  // ... definición de variables ...
  switch (instruction.opcode) {
    // ... otras instrucciones
    case INSTR_IDIV: // Dividir dos enteros
      {
      // idiv: divide los dos enteros de la pila
      // de operandos, y deja el resultado en el
      // mismo sitio.
      jint v1, v2;
      v1 = getCurrentOperandStack()->popInt();
      v2 = getCurrentOperandStack()->popInt();
      getCurrentOperandStack()->pushInt(v2 / v1);
      break;
      }
    // ... otras instrucciones
  } // switch
} // executeInstruction
```

Figura 5.9: Esquema de interpretación de `idiv` en una JVM

cación de la JVM original, y no es habitual tampoco durante la implementación de una JVM real. Como hemos dicho, una microinstrucción no es más que un paso en la ejecución de una instrucción, y en las implementaciones eso termina significando una línea de código adicional. Por ejemplo, en la figura 5.9 se muestra el posible código fuente (en C++) del intérprete de la instrucción `idiv` en la implementación de una JVM. Esta instrucción supone desapilar dos enteros de la pila de operandos del *frame* actual, dividirlos, y apilar de vuelta el resultado. En la figura vemos que eso lo conseguimos con tres líneas de código.

El problema es que esa información sobre lo que hay que hacer para ejecutar cada instrucción está incrustada en la implementación *de manera procedimental*, pero no es posible utilizarlo para saber si el estudiante está o no dando los pasos correctos. Necesitamos por lo tanto *conocimiento declarativo* gracias al cual el sistema "sea consciente" de los pasos de cada instrucción para poder utilizarlos en la fase de tutoría.

En concreto, la instrucción de división se disecciona en *cuatro* pasos, que requieren *tres microinstrucciones* diferentes. Los dos primeros pasos consisten en la que hemos llamado microinstrucción `pop`, el tercero es un `div`, y el cuarto un `push`.

El sistema, por tanto, necesita la descripción sobre qué microinstrucciones

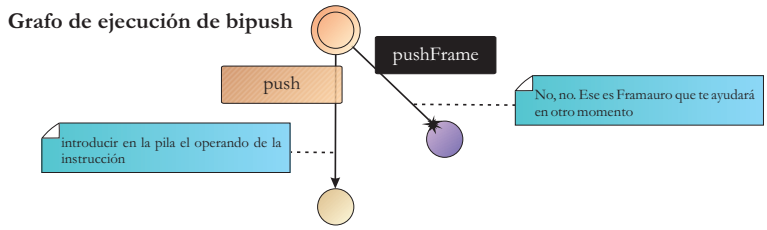


Figura 5.10: Grafo de ejecución de `bipush` con un camino erróneo

se deben realizar para ejecutar cada instrucción de la JVM. Este conocimiento se mantiene en lo que denominamos *grafos de ejecución*. Cada instrucción de la JVM tiene uno de estos grafos asociado. En ellos, cada nodo representa un estado de la ejecución de esa instrucción, mientras que las aristas son los pasos que hacen avanzar el proceso mediante una microinstrucción. Las aristas también guardan una explicación que indica por qué esa microinstrucción *es válida en ese momento*. Con esto tenemos que la jerarquía mantiene una descripción general de cada microinstrucción, y el grafo de ejecución almacena una explicación *contextualizada a la instrucción* donde se utiliza.

En la figura 5.10 aparece el grafo de ejecución de una instrucción sencilla (`bipush`), en la que únicamente se requiere la ejecución de una microinstrucción (`push`). Se aprecia que la arista en la que aparece está etiquetada con la explicación: "introducir en la pila el operando de la instrucción". Si el usuario intenta realizar otro paso, Javy utiliza ese texto y lo mezcla con una plantilla para construir su consejo: "Si yo fuera tú, intentaría *introducir en la pila el operando de la instrucción*".

En caso de que el estudiante siga confundiéndose, Javy busca el concepto de la microinstrucción válida en la jerarquía conceptual, recupera su campo *nombre* (en este ejemplo "apilar el valor", como aparece en la etiqueta asociada a él en la figura 5.5), y lo aplica a otra plantilla, para decir "Para *introducir en la pila el operando de la instrucción*, hay que *apilar el valor*".

Los grafos de ejecución, además, pueden guardar explícitamente caminos erróneos con explicaciones especializadas sobre los errores más comunes, de forma que Javy pueda proporcionar consejos. En la figura 5.10 se aprecia que `pushFrame` no es una microinstrucción válida, y, si se ejecuta, el agente diría "No, no. Ese es Framauro que te ayudará en otro momento"[5]. En ese caso, si el alumno se equivoca ejecutando esa microinstrucción concreta, en lugar de dar el consejo genérico construido con la plantilla se proporciona esta explicación más específica.

Aunque en la figura no se muestra, los grafos de ejecución contienen información adicional sobre aquellos objetos que se deben manipular. Por

[5]Framauro es un personaje auxiliar a través del cual el usuario manipula la pila de frames.

ejemplo, la instrucción `bipush` del ejemplo tiene *un operando* con el valor a apilar. Es necesario que ese requerimiento quede marcado de forma explícita en algún sitio; por otro lado, se necesitará una *coherencia* de forma que el sistema obtenga automáticamente ese operando y se lo haga llegar al avatar del jugador colocándolo automáticamente en su inventario.

Por su parte, la microinstrucción `push` (consistente en, efectivamente, apilar el valor en la pila de operandos) debe hacer referencia al objeto que hay que apilar. Esto repercute en un último aspecto que omitimos intencionadamente al hablar de la jerarquía conceptual. Aquellos conceptos que se encuentran dentro de la categoría de *instrucción* o *microinstrucción* poseen información adicional sobre su "prototipo", consistente en el número de objetos que reciben "de entrada" (manipulan) o generan "de salida" (es decir obtienen, como por ejemplo al extraer un dato de la pila de operandos).

La información sobre el prototipo se extrae de la jerarquía conceptual; los grafos de ejecución relacionan los objetos obtenidos en la ejecución de unas microinstrucciones con los utilizados en las posteriores, de modo que pueda asegurarse que el usuario está obteniendo y utilizando correctamente los diferentes valores intermedios. Por ejemplo, en la instrucción `idiv` mencionada antes, es necesario controlar si el primer operando extraído de la pila ("resultado" de la ejecución de la microinstrucción `pop`) será el dividendo o el divisor. La jerarquía conceptual tiene registrado que, efectivamente, `pop` origina la recogida de un valor nuevo, pero es el grafo de ejecución de `idiv` el que controla qué se hace con él en el resto de pasos.

Los grafos de ejecución se desarrollan con una tercera herramienta bautizada JaMOn (figura 5.11). El resultado se graba, como no podía ser de otra forma, en un XML cuya DTD aparece en la sección C.3.1. En la C.3.2 se muestra el XML del grafo de la figura 5.10. Igual que JaDe, durante la inicialización carga la jerarquía conceptual para conocer los conceptos asociados a las instrucciones y a las microinstrucciones, y garantizar la coherencia. Así, evita que una instrucción utilice microinstrucciones con las que no esté relacionado en la jerarquía y valida el uso de los parámetros de las instrucciones y microinstrucciones "declarados" también en la jerarquía. También permite añadir las explicaciones oportunas y los caminos no válidos. Debido a la fuerte dependencia entre la jerarquía conceptual y los grafos de ejecución, ambas herramientas (JaCo y JaMOn) terminaron siendo fusionadas en una única aplicación que pasó a llamarse JaCoMOn.

5.9. El análisis de la solución

Hemos dicho que el sistema controla continuamente los pasos que el alumno va realizando en el entorno para saber si está o no haciéndolo correctamente. Para eso, tenemos en realidad *dos niveles* de comprobación, pues estamos enseñando simultáneamente *dos* cosas. Por un lado, el objetivo último del

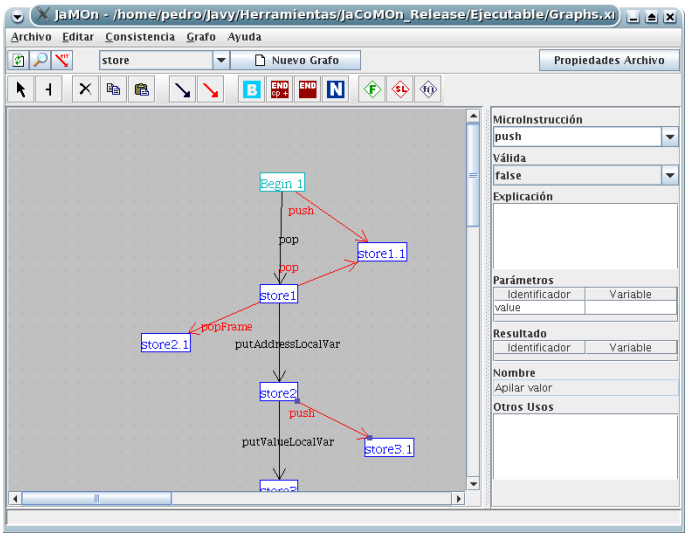


Figura 5.11: JaMOn (editor de grafos) en ejecución

sistema es que el alumno *compile* correctamente el código del ejercicio, pero esa compilación pasa en realidad por la *ejecución correcta* en el entorno. Es decir, en última instancia el sistema sólo puede monitorizar *la ejecución*, no *la compilación*. De nada le sirve a un alumno saber que tiene que utilizar la instrucción (de la JVM) `dup` si luego no es capaz de ejecutarla.

Por tanto, tenemos por un lado el "modelo" de la JVM controlado por el *simulador* de sus estructuras en la capa inferior. El alumno manipula ese simulador a través del entorno metafórico con el objetivo de ir ejecutando las instrucciones. A nivel de conocimiento del sistema, este aspecto educativo es cubierto por los *grafos de ejecución*.

Por otro lado tenemos el dominio "de alto nivel" consistente en el *proceso de traducción* del código. La enseñanza de este dominio de ámbito superior está respaldada por la base de ejercicios. Éstos se quedan al nivel de *instrucciones* de la JVM, que hacen las veces de punto de unión con el dominio inferior dado que los grafos de ejecución, que *se comparten* entre todos los ejercicios, comienzan a partir de ellas.

Como hemos dicho, la monitorización del alumno se hace a nivel de la ejecución en orden correcto *de las microinstrucciones*, y a partir de ahí es necesario averiguar si está ejecutando la *instrucción* actual propuesta por el ejercicio. Aparte de la necesidad de enlazar estos dos niveles también es necesario darse cuenta de que el estudiante en realidad *no* nos indica directamente la microinstrucción que ejecuta, sino que actúa en función de las *acciones primitivas* del entorno virtual.

Es decir, los grafos de ejecución sirven de enlace entre las instrucciones

de la JVM y la *ejecución* de cada una de ellas por parte del estudiante. Aunque añaden conocimiento declarativo que en otro caso quedaría oculto en la implementación, es necesario resaltar que el sistema mantiene codificado de manera procedimental el significado de cada microinstrucción *en el entorno*. Es decir, los grafos de ejecución se quedan al nivel de microinstrucciones (por ejemplo, `push`), pero el sistema ha de mantener la relación de cada una de ellas con la acción correspondiente sobre el entorno (por ejemplo "usar valor del inventario con la pila de operandos").

Para poder realizar la monitorización del alumno, el sistema debe ser capaz, por lo tanto, de convertir las acciones en el entorno a *microinstrucciones*. Para eso se ha desarrollado un módulo específico incrustado en la arquitectura software, que se encarga de cubrir el hueco entre ambos mundos (microinstrucciones y metáfora). Remitiéndonos a la figura 5.3 (página 182) que esquematizaba el funcionamiento de la aplicación, dicho módulo es el encargado de *filtrar* (de manera procedimental) los sucesos en el entorno virtual creado por el videojuego educativo y proporcionarle a la etapa de comprobación del ejercicio únicamente los sucesos significativos. Esta notificación se realiza de manera ya "tratada" de forma que se proporcionan directamente las *microinstrucciones* ejecutadas por el usuario.

Para evitar confusiones es interesante hacer a este respecto una matización. Ya se comentó que la jerarquía conceptual contiene también *conocimiento declarativo* sobre la metáfora para proporcionar ayuda al estudiante en caso de ser solicitada. Es decir, JAVY es capaz de *explicar* cómo realizar sobre el entorno una determinada microinstrucción. Este conocimiento declarativo, sin embargo, *no* es utilizado para realizar la traducción entre microinstrucciones y acciones primitivas en el entorno que necesitamos tener disponible en el programa. La razón principal es que la jerarquía conceptual está creada con el objetivo de proporcionar explicaciones al estudiante, no para *razonar* sobre ella.

Por su parte, el sentido inverso (microinstrucción → metáfora) también es necesario para aquellos casos en los que JAVY esté ejemplificando algo. Cuando es él quién resuelve un ejercicio, debe *realizar acciones* a partir del conocimiento declarativo mantenido en los grafos de ejecución.

Como se ha dicho, el modo en el que ambas traducciones se realiza queda enraizado en el código de la aplicación y los detalles están más allá del objetivo de este trabajo. Gómez Martín (2007) trata en detalle los aspectos de la *arquitectura software* de la aplicación, dedicando cierto esfuerzo precisamente a esta traducción.

En cualquier caso, el modo en el que se realiza la monitorización del estudiante es sencilla una vez que se traducen sus acciones en el entorno a microinstrucciones. El ejercicio recuperado posee su solución, por lo que el sistema sabe qué instrucción de la JVM es la siguiente que debe ejecutarse (compilarse). Además, con el grafo de ejecución correspondiente sabe cual es

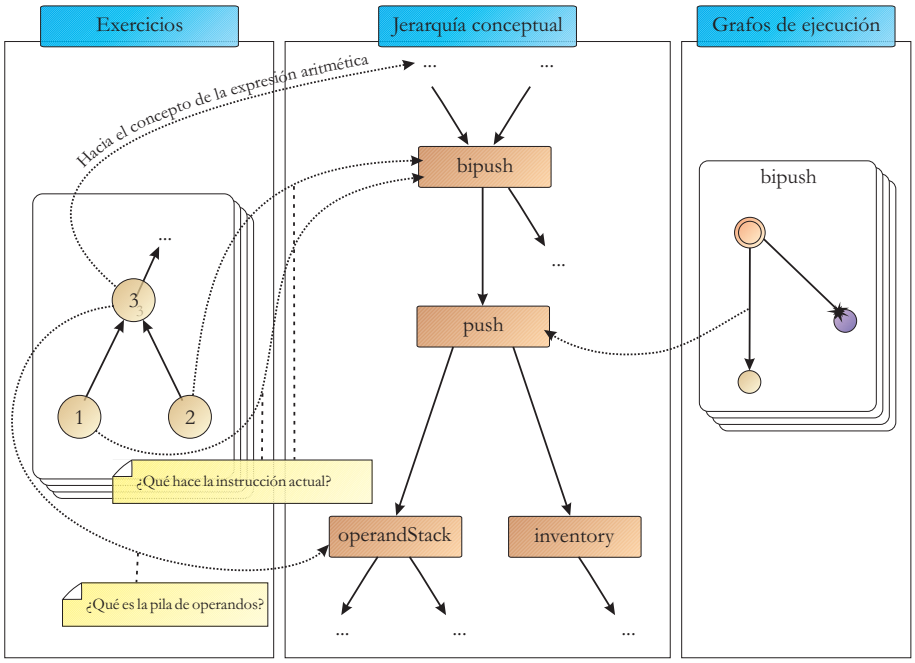


Figura 5.12: Visión general del conocimiento de JV^{2}M

la microinstrucción que el alumno deberá realizar. Si el estudiante hace una acción sobre el entorno que se corresponde con una microinstrucción diferente, el hecho es detectado inmediatamente. Ya dijimos que no permitimos que el alumno se desvíe del buen camino, por lo que ante cualquier error se le detendrá y se le proporcionará una pista.

5.10. Comunicación con el estudiante

Hasta ahora hemos visto los tres tipos de conocimiento mantenidos por el sistema (jerarquía conceptual, ejercicios y grafos de ejecución), todos ellos útiles para proporcionar información al usuario gracias a los textos en lenguaje natural empotrados en ellos. Se ha esbozado, además, que añadimos *preguntas* que el alumno puede hacer una vez que el agente pedagógico le ha proporcionado una determinada explicación. Esto indica que el sistema mantiene un *diálogo* con el alumno al estilo de los juegos de aventura.

Para mejorar esta interacción, los tres bloques de conocimiento analizados *no* están aislados, sino relacionados entre sí utilizando enlaces adicionales. Por ejemplo, los bloques de código de los ejercicios suelen estar enlazados con nodos de la jerarquía conceptual, de forma que los bloques pequeños se relacionan con el concepto de la instrucción de la JVM que contienen, y los grandes se relacionan con los conceptos de compilación. Del mismo modo,

Figura 5.13: JV²M en ejecución

las aristas de los grafos de ejecución están relacionadas con el concepto de la microinstrucción asociada en la jerarquía conceptual.

Todos estos enlaces adicionales también están etiquetados con preguntas que el usuario puede hacer cuando el estado de la conversación está localizado en el nodo del que sale la arista. La figura 5.12 muestra la estructura general.

Para comprender como todas las piezas aquí descritas entran en juego durante la resolución de un ejercicio describiremos como ejemplo un posible escenario de resolución del ejercicio de la figura 5.7[6].

Utilizando el teclado, es posible consultar el código Java que hay que compilar. En este ejemplo, al principio del ejercicio, se debe ejecutar la instrucción `bipush 5`. Dado que ésta está relacionada con el número `5` de la expresión del código Java, éste aparece destacado al principio del ejercicio (ver figura 5.13).

La instrucción `bipush 5` tiene un parámetro que se añade automáticamente al inventario del usuario como un objeto virtual. Imaginemos que el usuario lo selecciona e intenta dárselo a Framauro. Esa acción se corresponde con la microinstrucción `pushFrame`. Como mostrábamos en el grafo de ejecución en la figura 5.10, JAVY interpreta este paso como inválido y da una explicación adaptada a este error, obteniéndola del grafo: "No, no. Ese es Framauro que te ayudará en otro momento". Ante este error, supongamos que el usuario se rinde, y pregunta a JAVY. En el primer paso o ciclo de la conversación, el agente permite al usuario hacer siempre las mismas

[6]Video disponible en `http://gaia.fdi.ucm.es/grupo/projects/javy/`

preguntas generales:

- ¿Qué se supone que tengo que hacer ahora?
- ¿Por qué tengo que hacer esa instrucción?
- Déjalo. Ya me las apaño.

La última frase siempre estará disponible, para permitir al usuario terminar el diálogo. Las otras dos aparecen siempre que se inicia una conversación. Sin embargo, cada vez enlazarán a diferentes partes del conocimiento, dependiendo del contexto.

En concreto, la primera frase se relaciona con la microinstrucción que tiene que ejecutarse a continuación. Si se selecciona, JAVY utiliza la descripción asociada y la plantilla descrita en la sección 5.8 para decirle al usuario: "Para *introducir en la pila el operando de la instrucción* hay que *apilar el valor*".

Después de esto, el agente construye una nueva pregunta válida, utilizando otra plantilla y el nombre de la microinstrucción de la jerarquía conceptual: "Cuéntame algo más sobre eso de *apilar el valor*". Esta frase enlaza con el concepto de la jerarquía conceptual de esa microinstrucción.

Si el estudiante la selecciona, JAVY consulta su fuente de conocimiento para dar la explicación asociada (que, como muestra la figura 5.5 de la página 187, es: "Hay que hacer un push. Para eso, lleva el operando del inventario a la pila de operandos"). En ese momento, el alumno podrá ir preguntando por los conceptos más profundos de la jerarquía conceptual, según se ha explicado en la sección 5.6.

Esto debe dejar claro que la primera pregunta disponible al principio de toda conversación proporciona información sobre los conceptos que hacen referencia a instrucciones de la JVM y sus estructuras, y que dependerá de la instrucción y microinstrucción actual. Por ejemplo, no se hablará nada de orientación a objetos, si se está en `bipush 5`.

Por otro lado, la segunda pregunta posible al iniciar la conversación ("¿Por qué tengo que hacer esta instrucción?") enlaza siempre con la información del ejercicio, más concretamente con el bloque más profundo que contiene la instrucción actual. Imaginemos que el usuario ya ha ejecutado la primera instrucción y se encara a la segunda, `bipush 3`. Si el estudiante se acerca al agente para iniciar la conversación, ahora la segunda pregunta enlazará con el bloque más profundo con esa instrucción, es decir, el etiquetado en la figura 5.7 con el número 2. Cuando el estudiante la selecciona, JAVY utiliza la información adjunta: "Ya está apilado el primer operando (el 5). Ahora hay que apilar el segundo, el 3".

En ese momento, el usuario tiene disponibles dos preguntas. La primera enlaza con el *bloque padre* del código, que aparece en la figura 5.7 (etiquetado

con un 3). Esta conexión permite al estudiante *navegar* en el árbol de bloques, consiguiendo información sobre el proceso de compilación, es decir, sobre los *conceptos de alto nivel*. La segunda pregunta disponible enlaza con la *jerarquía conceptual* utilizando la conexión que se muestra en la figura 5.12, por lo que la pregunta es "¿Qué hace la instrucción actual?" que termina en el concepto `bipush`.

Conclusiones

En el capítulo anterior planteamos un modo de fusionar los videojuegos y las aplicaciones educativas utilizando como eje conductor la dualidad ejercicio – nivel de juego. En este capítulo hemos descrito una instanciación de esas ideas, JV^{2}M, en el que se integra dentro de un videojuego una aplicación educativa que enseña, de manera intuitiva, el proceso de compilación de código Java en código objeto de la JVM.

El género del videojuego utilizado es de aventura, donde la labor principal del jugador es manipular el entorno para ir realizando acciones que le acerquen al final. Ese entorno consiste en realidad en una metafórica máquina virtual donde *ejecuta* el código del ejercicio que debe compilar *al vuelo*. Para dar consistencia, existe una *historia de fondo* que proporciona sentido a la tarea, y que queda escrita en el apéndice B.

El sistema monitoriza al estudiante a lo largo de todo el episodio de aprendizaje, controlando por dónde va, y proporcionándole información adaptada a la situación concreta. Para eso, el entorno virtual está habitado por un agente pedagógico, JAVY, que hace las veces de *entidad visible* del componente pedagógico del sistema. JAVY proporciona ayuda bajo demanda, pero también actúa de manera proactiva cuando observa que el estudiante se desvía de la solución correcta.

Todo lo anterior pone de manifiesto que, efectivamente, el sistema cumple los objetivos propuestos en el capítulo anterior. Sin embargo, sufre algunas deficiencias. Por desgracia, es arriesgado defender que, efectivamente, el videojuego *es divertido*. Cuando los ejercicios se van haciendo más complicados, la repetición de tareas lentas, tediosas y ya conocidas hace que la idea de tener que abordar un ejercicio más resulte poco atractiva. Para solventar este problema la historia de fondo plantea un marco que justifica la aparición de personajes auxiliares que alivian un poco el tedio de la repetición.

En cualquier caso, el problema está demasiado arraigado en el funcionamiento del programa como para que esto lo solucione completamente. Al fin y al cabo, como vimos en la sección 3.3.1, el género de aventura se centra en la *narración* más que en cualquier otra cosa. Los juegos tradicionales de aventura disponen de una *historia que evoluciona*. Aparecen personajes nuevos, ocurren cosas no previstas, los objetivos del jugador fluctúan, etcétera. Además, para adornar el juego, a menudo se añade una pizca de humor a través

de algunos de los actores secundarios que proporcionan diálogos hilarantes que han pasado a formar parte del folclore de este tipo de juegos.

Nuestro uso del género de los juegos de aventura se debe más *al modelo de interacción* que a la importancia de la narración. De hecho, la historia de fondo *apenas evoluciona*, el objetivo del usuario es siempre el mismo y los personajes no cambian nunca. Por tanto, los ingredientes principales de los juegos de aventura están ausentes aquí. El motivo principal es que nosotros *generamos* los niveles de juego automáticamente a partir de los ejercicios. En los juegos de aventura son *guionistas* los que se preocupan de realizar la narración, avanzando la historia, creando personajes y escribiendo los diálogos con sus pinceladas de humor. Conseguir hacer esto *de manera automática* es un incipiente área de investigación muy amplia que debe lidiar con la creatividad, las técnicas artísticas y la coherencia del resultado. Dada la complejidad del problema, nunca hemos llegado a abordarlo en profundidad más allá de lo que describimos en Peinado Gil et al. (2005).

Por otro lado, las técnicas de Inteligencia Artificial utilizadas son bastante manuales. Esto impide hacer uso de la batería de herramientas que este campo de la ciencia nos proporciona, lo que repercute negativamente en la cantidad de características incluidas. En concreto, el sistema *prohíbe* al estudiante desviarse del buen camino en la resolución del ejercicio, lo que coarta su libertad y su posibilidad de exploración. Esto tiene repercusiones tanto en la parte lúdica (normalmente los juegos de aventura permiten gran cantidad de pruebas hasta dar con la acción correcta) como en la parte pedagógica al desaprovechar la posibilidad del aprendizaje por descubrimiento. En lo referente a la representación del conocimiento, estamos haciendo uso de técnicas de CBR "puro", en el que la base de casos se encuentra poblada sin utilizar conocimiento del dominio adicional. El uso de KI-CBR que esgrimíamos en el capítulo 4 ha quedado fuera de esta instanciación.

En el próximo capítulo...

En el próximo capítulo se expone una nueva instanciación del modelo de aplicaciones educativas planteado en el capítulo anterior. Se utiliza el mismo dominio objetivo en la parte pedagógica, pero se hace uso de un *género* de juego diferente. Esto pretende solventar los problemas respecto al entretenimiento descritos en la sección anterior.

Además, realiza un desplazamiento en las técnicas de Inteligencia Artificial utilizadas, para poder utilizar la "artillería pesada" que ésta nos proporciona. Aunque, naturalmente, se siguen teniendo en mente las cinco etapas de ejecución descritas en el capítulo anterior, a nivel de implementación el funcionamiento es bastante diferente respecto al descrito aquí.

Capítulo 6

JV^{2}M 2: Realización del modelo basada en casos ricos en conocimiento

– [...]; y desta verdad te pudiera traer tantos ejemplos, que te cansaran.

Don Quijote de la Mancha II,
Miguel de Cervantes

Resumen: En este capítulo se describe una segunda instanciación del modelo de funcionamiento de videojuegos educativos que se especificó en el capítulo 4. En este caso se ha realizado un cambio radical en el *diseño* del videojuego, dotándole de un mayor dinamismo para evitar la repetición de tareas monótonas. Además, la parte de tutoría ha sido replanteada, siguiendo una aproximación mucho más académica en el modo de especificar el conocimiento. El uso de razonamiento basado en casos con soporte de ontologías para añadir conocimiento terminológico sobre el dominio brilla aquí con especial esplendor.

6.1. Introducción

El modelo general de fusión de videojuegos y aplicaciones educativas que se planteaba en el capítulo 4 se basaba en la idea de aglutinar un nivel de juego con un ejercicio. En el capítulo anterior se describió una particularización de dicho modelo, en una aplicación que tenía como objetivo la enseñanza informal del proceso de traducción de código Java a código de la JVM.

En este capítulo planteamos una nueva instanciación del modelo (Gómez Martín et al., 2006a). El sistema descrito enseña el mismo dominio de la traducción de código Java, por lo que puede considerarse una evolución del anterior; de hecho el nombre con el que se le ha bautizado, JV^{2}M 2, hace un fiel tributo a sus orígenes.

No debería, sin embargo, verse como una mera revisión del sistema anterior, sino como una remodelación profunda que ha afectado a muchos niveles. El *diseño* del videojuego ha sido totalmente reescrito, afectando *profundamente* al modo en el que la parte lúdica y la parte educativa se entrelazan. En concreto, se ha abandonado el género de aventura para pasar a un juego con una alta carga de acción. Esto repercute en el objetivo del jugador y también en el modo que éste tiene de comunicarse con el sistema.

A nivel de implementación de la parte educativa se ha dado un giro completo hacia representaciones más académicas del conocimiento. Demostramos aquí las bondades del razonamiento basado en casos rico en conocimiento del que se habló en el capítulo 4, que es aplicado de manera independiente (pero coordinada) en varias de las etapas del ciclo de ejecución.

Esto significa que hay dos grandes puntos de cambio: el género del videojuego, y el modo de representar el conocimiento. Aparte de mejorar la primera versión, esta nueva implementación vuelve a poner a prueba, una vez más, el modelo general del capítulo 4. Veremos que éste sigue encajando sin dificultad. Es necesario recalcar que aunque las dos instanciaciones que se plantean de dicho modelo se utilizan para enseñar el mismo dominio, también se adapta, naturalmente, a cualquier otro. Mantener la enseñanza del proceso de traducción nos evita, sencillamente, preocuparnos por buscar expertos en otro área.

En la siguiente sección se realiza un análisis crítico de la primera versión de JV^{2}M (que llamaremos JV^{2}M 1 en lo sucesivo cuando pueda haber ambigüedad). Las conclusiones extraídas sirven, en realidad, como una lista de "requisitos" que la siguiente versión deberá intentar cumplir. Posteriormente se describe el nuevo diseño del videojuego que busca incrementar la diversión, pues la versión anterior terminaba siendo repetitiva y un tanto lenta. Esto da una idea sobre la visión que el *usuario* tiene del sistema, que se completa con la visión general a nivel de funcionamiento interno proporcionada por la sección 6.4. El resto de apartados van diseccionando cada uno de los tipos de conocimiento de los que el sistema hace uso y que tienen una relación directa con las etapas de ejecución que ya se describieron en el capítulo 4.

6.2. Análisis crítico de JV^{2}M 1

El sistema JV^{2}M descrito en el capítulo anterior pone a prueba la idea de igualar un ejercicio (parte pedagógica) con un nivel (parte lúdica) para formar un videojuego educativo. Además, alrededor de esta unión se colocan

con éxito el resto de etapas en el ciclo de *aprender haciendo*, de modo que es el sistema el que elige el siguiente ejercicio a practicar (nivel a jugar), y hace las veces de guía cuando el estudiante se confunde.

Sin embargo, aunque sirve como una encarnación de ejemplo de las ideas descritas en el capítulo 4, muestra varias deficiencias que nos proponemos solucionar con la siguiente versión del sistema expuesta a continuación. En primer lugar, el *diseño del juego* está demasiado cerca de una aplicación educativa. Es cierto que lo que queremos enseñar es *cómo traducir* código Java en código de la JVM, y a pesar de eso el objetivo del juego lo hemos desplazado hacia *ejecutar* dicho código resultante. Esto pretende añadir actividad (movimiento e interacción) a las mecánicas que, de otra manera quedarían muy cercanas a escribir código. Sin embargo, este modo de proceder supone que, al cabo de un tiempo, la repetición de las mismas tareas una y otra vez termine siendo demasiado pesada. La causa última de este problema estriba en que al forzar a *ejecutar* el resultado, ralentizamos la resolución de los ejercicios. Esto no es demasiado problema al principio, cuando éstos son sencillos y de hecho la ejecución facilita la adquisición de conocimiento sobre la JVM. Sin embargo, superadas las primeras interacciones la *carga* de tener que dar muchos pasos para realizar algo conocido y (ya) sencillo añade mucho trabajo poco motivante que aburre al estudiante.

Para solucionarlo, se añadieron a la historia de fondo los ayudantes (encarnados en los campesinos). Si bien esto puede ayudar a aliviar un poco la repetición, siguen faltando mecánicas de juego que mantengan el interés. En concreto, decidimos utilizar como género inspirador los juegos de aventura. Pero de ellos importamos únicamente el modo de interacción con el sistema. Esto aplana la curva de aprendizaje *del interfaz* de la aplicación, pero no añade diversión "per se". En concreto, el peso principal de los juegos de aventura gira alrededor *de la narración*, creando una historia que evoluciona en la que el *jugador* es el protagonista. Nuestra historia, sin embargo, apenas tiene interés más allá de servir de *marco de fondo* para dar un mínimo sentido al programa. No hay una narración interesante y cambiante una vez que el juego comienza, por lo que la base principal sobre la que se asienta el género (y que es la que desata la curiosidad y anima a seguir jugando) desaparece.

Por su parte, en la labor del estudiante en lo referente al *dominio* que se enseña, existe un problema no siempre obvio que viene implícito al agrupar *traducción* y *ejecución*: el contador del programa. No es hasta la aparición de ejercicios con *instrucciones de salto* (en el código objeto) cuando el problema se hace patente. Para ponerlo de manifiesto, consideremos el código de la figura 6.1, en el que aparece un pequeño código en Java junto con su versión compilada para la JVM que el usuario debe ser capaz de generar. Por claridad, se han dejado intencionadamente lineas en blanco en el código fuente para sincronizar las instrucciones de la JVM con las sentencias Java

```
int cont = 0;              //  0: iconst_0
                           //  1: istore_0
int aux = 4;               //  2: iconst_4
                           //  3: istore_1
while(cont < 10) {         //  4: iload_0
                           //  5: bipush 10
                           //  7: if_icmpge 19
   aux++;                  // 10: iinc 1,1
   ++cont;                 // 13: iinc 0,1
}                          // 16: goto 4
                           // 19: return
```

Figura 6.1: Código que muestra el problema del contador de programa

a las que corresponden.

Como la compilación se realiza *a la vez* que la ejecución, el contador del programa *lo controla el usuario*. Al llegar a la instrucción número 7 (`if_icmpge 19`[1]) es necesario realizar la comprobación. En el primer momento no será cierto, y habrá que ejecutar (compilar) los dos incrementos. Finalmente se llega a la instrucción 16 (`goto 4`). Pero esta instrucción ¿realmente debe "compilarse"? En realidad al estar *ejecutándose* el código, tras el segundo incremento el usuario deberá *repetir* el bucle completo. Es decir, el estudiante no realiza una compilación real, sino más bien la ejecución "desenrollada" del código. En concreto, al *encarnar* el contador de programa, tras el segundo incremento se verá obligado a *ejecutar* de nuevo la comprobación, y repetir el bucle las 10 veces. De la misma forma, cuando la condición de la instrucción 7 se haga cierta, al ser el propio usuario quién representa el contador del programa resultará difícil pensar que se *realiza* un salto, pues, sencillamente, pasará a ejecutar el `return` sin ser consciente de dicho salto o, incluso, de la numeración de las instrucciones.

En lo que se refiere a la parte pedagógica o de enseñanza asistida, JV^{2}M 1 tiene también algunas deficiencias. La primera es que no permite que el alumno se equivoque. En cuanto comete un fallo, el sistema impide la ejecución de la operación inválida y se le proporciona una pista. Aunque esta realimentación inmediata es defendida por muchos, por desgracia dinamita el carácter exploratorio y de descubrimiento de muchos juegos que, además, era uno de los ingredientes principales de los videojuegos de aventura. Por otro lado,

[1] Esta instrucción significa que si el valor de la cima de la pila de operandos del *frame* actual es mayor o igual que el que está debajo de él, hay que saltar a la instrucción `19`.

también puede dañar el aspecto pedagógico al dejar pasar la oportunidad de que el alumno aprenda de sus propios fallos ("si cierras la puerta a todos los errores, también la verdad se quedará fuera", decía Rabindranath Tagore[2]).

Además, las explicaciones disponibles para *errores típicos* lo están a nivel de instrucciones *de la JVM*, no del proceso de compilación. Es decir, si el alumno realiza una *microinstrucción* inválida, y tenemos una explicación adaptada a ese error, se la proporcionamos (en lugar de darle una pista genérica). Pero en el dominio de alto nivel (compilación) no tenemos explicaciones para los errores. De hecho, únicamente soportamos un modo de compilar el código fuente, cuando algunas construcciones pueden traducirse de múltiples formas. Esto es especialmente problemático cuando el alumno va a empezar una instrucción nueva. Supongamos que el alumno decide que quiere empezar a ejecutar `dup`, pero la solución almacenada junto con el ejercicio mantiene que la siguiente instrucción correcta es un `bipush 10`. El alumno comenzará ejecutando la primera microinstrucción de `dup`, en concreto, extraer el valor de la cima de la pila de operandos. Sin embargo, el sistema está esperando que haga justo lo contrario (primera microinstrucción de `bipush`), por lo que le parará y le dirá "Si yo fuera tú intentaría introducir en la pila el operando de la instrucción" (figura 5.10), algo que no aporta al estudiante demasiada información. Podemos decir que el sistema ha fallado al identificar el *genotipo* (causa) del error a partir de su *fenotipo* (manifestación).

Si tuviéramos suerte, podríamos tener en el grafo de ejecución de la instrucción `bipush` una explicación adaptada al error de ejecutar la *microinstrucción* de desapilar un valor. En ese caso, la explicación estará, irremediablemente, enfocada a la idea de que el alumno *quiere* ejecutar un `bipush` pero lo está haciendo mal. Sin embargo, el estado mental del alumno es que quiere ejecutar un `dup` y tiene la firme creencia de que lo está haciendo bien, por lo que la ayuda seguiría siendo inútil.

En principio parece que para solventar este problema podríamos mantener en cada ejercicio no sólo una, sino varias formas de compilarlo. Pero, aparte de un incremento significativo del trabajo de autoría, no conseguiríamos nada. Esto es debido a que es el sistema quién *descodifica* la siguiente instrucción a ejecutar. Es decir, el sistema decide que, en su opinión, lo siguiente a ejecutar es `bipush 10` y debido a eso coloca en el inventario del jugador el operando de dicha instrucción (el `10`). Como el alumno *no tiene forma* de "crear" o "destruir" valores dentro del sistema, no puede hacer nada para ejecutar una instrucción diferente a la prevista por la solución del ejercicio. Esto en realidad tiene una parte positiva, pues el estudiante recibe cierta información al consultar lo que ha obtenido en el inventario como resultado de la descodificación (automática) de la siguiente instrucción esperada. Es decir, si se encuentra un `10`, irremediablemente sabrá que la siguiente instrucción a ejecutar *no* es un `dup` (que no tiene operandos).

[2]Escritor bengalí y primer asiático en recibir el Premio Nobel de Literatura, en 1.913.

Por último, la autoría de ejercicios es bastante tediosa. Incluso con el apoyo de la herramienta (JaDe, figura 5.8) que invoca a `javac` y crea una estructura inicial relacionando código fuente y código objeto, construir la estructura de bloques arbórea y asociar explicaciones a cada uno de ellos es repetitivo, aburrido y propenso a errores. Al fin y al cabo, una expresión aritmética se compila de una determinada manera siempre por la misma razón, y a pesar de eso la creación de ejercicios exige que se anote y se haga explícita esa razón continuamente, incluso aunque la probabilidad de que el alumno la requiera (en ejercicios avanzados) sea mínima. La razón de fondo de este problema estriba en que estamos utilizando CBR muy sencillo, apoyado en muy poco conocimiento adicional del dominio que enriquezca la información que contienen. No hay que olvidar que la jerarquía conceptual añade información de otra índole que *no* es directamente utilizable *desde dentro* de los casos (ejercicios con sus soluciones). Por culpa de esto, los ejercicios se ven obligados a mantener mucha información repetida una y otra vez a lo largo de toda la base de casos.

Es necesario hacer un inciso aquí, recalcando que todos los problemas descritos *no son intrínsecos* de la idea general descrita en el capítulo 4 sobre el modo de fusionar aplicaciones educativas con videojuegos. JV^{2}M 1 ejemplifica perfectamente el modelo de aplicación que estamos buscando, proporcionando un ejemplo completo. Sin embargo, tiene dos claros puntos de mejora:

1. *El diseño del juego*: las mecánicas de juego utilizadas distan de ser las más entretenidas que uno querría encontrarse. De hecho, tenemos "juego" más bien debido al *modo de interacción*, no a los ingredientes lúdicos existentes. Esto añade *familiaridad* en el uso, ahorrando tiempo para el aprendizaje básico sobre cómo desenvolverse en el entorno, pero no necesariamente *diversión*.

2. *El modo de implementación de la parte educativa*: la integración de los ejercicios con los niveles trae consigo deficiencias en algunos casos. En concreto, a pesar de requerir gran labor de autoría, no siempre es capaz de proporcionar la información que el estudiante necesitaría.

En este capítulo planteamos un nuevo videojuego educativo para enseñar el mismo dominio, al que llamamos JV^{2}M 2. Se ha modificado radicalmente el *diseño* del videojuego, cambiando incluso el género subyacente. También se ha revisado la forma en la que se mantienen los ejercicios y se proporciona información al estudiante, realizando un desplazamiento hacia el KI-CBR. Es necesario recalcar que se siguen manteniendo por debajo las ideas expuestas en el capítulo 4, lo que pone de manifiesto su validez en diferentes videojuegos, con distintas ambiciones tanto en el lado de lúdico como en el lado educativo.

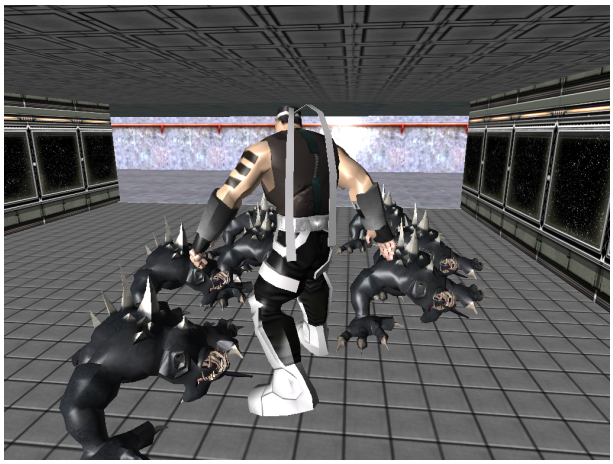


Figura 6.2: Captura de JV^{2}M 2: Portarecursos

6.3. La metáfora

El primer objetivo es mejorar el *diseño* del juego para revitalizar la parte lúdica de la aplicación. Como ya se ha dicho, la idea básica sobre la que se construye la experiencia de juego se mantiene: cada nivel es un ejercicio que pone a prueba los conocimientos de traducción de código fuente Java a código de la JVM. Sin embargo, para agilizar esta idea básica se plantean tres mecanismos:

1. *Acción*: se abandona el género de aventura y se añaden ingredientes de los juegos de acción, en los que, como decíamos en la sección 3.3.1, la jugabilidad se basa en la realización de acciones rápidas en tiempo real. Con esta idea, no será suficiente con dominar los conceptos del dominio que se enseña, sino que el jugador deberá demostrar una cierta habilidad en el uso del teclado y el ratón.

2. *Traducción en lugar de ejecución*: el usuario deja de tener que ejecutar el código que compila, y en su lugar sencillamente *lo escribe*. Esto elimina el tedio de la repetición de pasos casi insignificantes ya conocidos, y solventa, como veremos, el problema de las instrucciones de salto.

3. *Competición*: la competición es una de las piezas claves tradicionales en los videojuegos, a veces luchando contra otro jugador o, sencillamente, contra el tiempo. En esta versión del sistema el juego se convierte en una carrera contra un oponente (ya sea humano o controlado por ordenador) o contra el crono.

En la primera versión de la aplicación la decodificación de las instrucciones se realizaba de manera automática, de modo que los operandos que

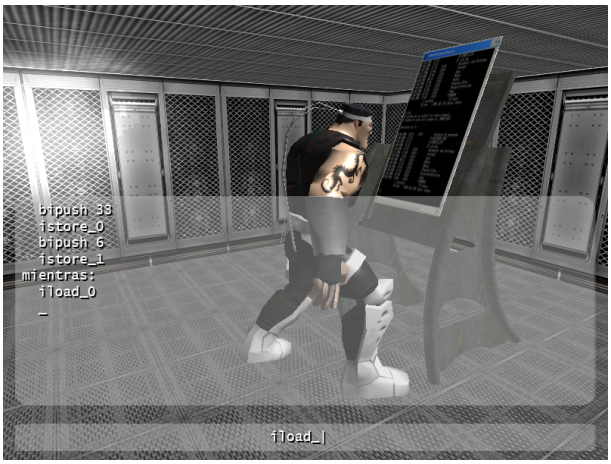


Figura 6.3: Captura de JV^{2}M 2: Terminal

el usuario necesitaba para la ejecución estaban disponibles sin ningún tipo de esfuerzo. La revisión de este modo de proceder es lo que ha convertido a JV^{2}M 2 en un juego de acción. El jugador debe *luchar* por conseguir los operandos (ahora denominados recursos) robándoselos a unos nuevos personajes a los que llamamos *portarecursos*. Para eso, es necesario "cazarlos" a lo largo y ancho del entorno virtual, sin saber, además, a ciencia cierta que porta cada uno (figura 6.2).

El siguiente cambio significativo es que olvidamos la ejecución manual de las instrucciones de la JVM a través de los pequeños pasos de las microinstrucciones. Dado que éste era uno de los motivos principales de tedio a medio plazo, ha sido eliminado completamente. De esa forma, el estudiante directamente *escribe* en un terminal (figura 6.3) las instrucciones que quiere añadir al código compilado. Para poder escribir una determinada instrucción (por ejemplo `bipush 10`) es necesario que el jugador *disponga* del "recurso" `10`, que deberá haber sido "robado" a alguno de los portarecursos que vagabundean por el entorno.

En realidad, la ejecución manual tenía en JV^{2}M 1 un sentido pedagógico, pues permitía comprender la estructura de la JVM y servía para aprender el significado de cada una de sus instrucciones. Al eliminar esta ejecución, es necesario buscar algún modo de que el alumno consiga ese conocimiento por otro lado. El entorno *sigue* representando una metafórica máquina virtual, en la que las diferentes estructuras internas de una JVM aparecen representadas con diferentes elementos (que, ahora, ya no son directamente manipulables por el estudiante). Por tanto, la ejecución de las instrucciones escritas en el terminal *afectan* al entorno, y es posible percibir qué cambios originan.

Para terminar de añadir diversión al juego, se añade un ingrediente de *competición* al luchar o bien contra el tiempo, o bien contra un oponente (humano o controlado por el ordenador). En este último caso ambos participantes deben compilar *el mismo ejercicio*. Con esto se añade una pizca de estrategia porque los recursos *se comparten*. Esto significa que para ganar no sólo es necesario conocer la forma de compilar el código fuente, sino que también es necesario darse más prisa que el contrario en encontrar los recursos necesarios. Esto añade además de manera intrínseca en el diseño una característica positiva en la parte educativa. En concreto, si existen varias formas de compilar correctamente el ejercicio, se incita *por diseño* a los jugadores a pensar aquella que menos recursos necesite para acortar su búsqueda y aventajarse respecto al contrario.

Para que todos estos cambios tengan sentido, hay que eliminar también una de las restricciones que en JV^{2}M 1 limitaba la diversión: la libertad de equivocarse. Como dijimos, en la primera versión *no* permitíamos al estudiante salirse del buen camino. Este modo de proceder seguía la máxima de que la información tiende a recordarse más si se proporciona en el momento de necesitarse. Pero, además, simplificaba el control de la situación de la resolución por parte del sistema pues en ningún caso tenía que controlar un encadenamiento de errores. Esto, sin embargo, va en contra de lo que se suele encontrar en los videojuegos, donde prima la libertad y el descubrimiento. Ahora sí vamos a permitir que un estudiante se equivoque, y aun así persevere en su error casi sin limitación. Cuando el usuario es consciente de que está yendo por un camino incorrecto, tiene que solicitar al sistema que se deshagan sus pasos para recuperar la última situación válida de la resolución. Estos instantes anteriores (seguros) a los que volver son semejantes a los *puntos de guardado automático* que muchos videojuegos tienen y que ya se mencionaron en la sección 4.3.

Naturalmente, esto complica la implementación, pero eso no debería ser algo que se tenga demasiado en mente durante la fase de diseño. Aparte de la diversión e incertidumbre que se añade con esta libertad, por debajo también tiene sentido pedagógico. Cometer equivocaciones al resolver el ejercicio tiene repercusiones negativas en el desarrollo del juego porque *se pierde tiempo*. En un entorno de competición eso resulta bastante negativo al dificultar la posibilidad de ganar. Por tanto, algo perjudicial a nivel educativo (equivocarse) también lo es en la parte lúdica, lo que demuestra una buena integración de ambas componentes en el diseño.

Los jugadores podrían abusar del sistema para evitar perder el tiempo ante errores si inmediatamente después de ejecutar una instrucción solicitaran volver al último estado correcto. Si el sistema aún considera que el estado actual es correcto no se producirá ningún cambio, y si el jugador acaba de cometer un error, el sistema lo deshará impidiendo en gran medida la pérdida de tiempo por seguir un camino que no lleva a ningún sitio positivo.

Para evitar este modo de abuso, la vuelta a un punto correcto anterior *penaliza* quitando tiempo disponible en las partidas contra el reloj, o añadiendo tiempo al total como castigo. La penalización será mayor cuanto *menos* se retroceda, de modo que quede balanceada con el tiempo perdido por el propio error. Después de todo, un jugador que lleva dos minutos siguiendo un camino incorrecto ya ha sido castigado lo suficiente como para continuar ensañándose con él. Sin embargo, si el estudiante pide un "deshacer" de manera muy temprana, hay posibilidades de que esté intentando engañar al sistema y se le penalizará más. Naturalmente, es necesario *equilibrar* todos estos valores para que la jugabilidad no se vea perjudicada.

Del mismo modo, el sistema también será capaz de proporcionar ayuda. Para evitar también un abuso aquí, la obtención de información supondrá un coste en tiempo. Esto añade de manera implícita una decisión de diseño que tiene repercusiones importantes en la parte pedagógica y en la implementación: el sistema *ya no es proactivo*, es decir no proporcionará información por propia iniciativa. De hacerlo, estaría dando información "gratis", pudiendo producirse agravios comparativos con el oponente. Esta decisión es, además, coherente con la permisibilidad que la aplicación tiene ahora frente a los errores de los jugadores.

Para dar sentido a todo esto, se puede utilizar una gran cantidad de historias de fondo. En concreto, nosotros ambientamos el juego en una nave espacial enemiga que hay que sabotear activando bombas para abandonarlas allí y que exploten. La nave está dividida en *plantas*, cada una representando un *registro de activación*, de modo que la ejecución de instrucciones de invocación a métodos activan ascensores para subir a plantas superiores y las instrucciones de vuelta permiten volver a bajar. En cada planta se dispone de los recursos necesarios para compilar el método asociado a ese registro de activación (Gómez Martín et al., 2006c).

La evolución entre las dos versiones de la aplicación pone de manifiesto la importancia de una buena metáfora en el diseño del juego. Puede verse una comparación entre ambas en Gómez Martín et al. (2007b).

6.4. JV^{2}M 2: el sistema a vista de pájaro

Como hicimos en el capítulo anterior, antes de entrar en los diferentes detalles del sistema se dará aquí una descripción de alto nivel sobre su funcionamiento general. Para eso, utilizaremos como hilo conductor las cinco etapas que se describieron en el capítulo 4 en lo que se refiere al ciclo de ejecución del sistema.

La primera fase de dicho esquema era la elección de los conceptos que el alumno debe practicar. Es decir, en función del conocimiento que el sistema considera que tiene el estudiante, se toma la decisión sobre el siguiente episo-

dio de aprendizaje con la selección de los conceptos del dominio a practicar.

Las ideas de esta etapa no han cambiado en exceso respecto a la instanciación anterior. En concreto, el sistema mantiene información sobre los *conceptos del dominio* como ocurría antes. También recibe una secuencia creada por un experto sobre el mejor orden de enseñarlos, que, junto con un modelo del estudiante basado en plantillas, son los ingredientes necesarios para tomar la decisión de lo siguiente a practicar.

La diferencia principal estriba en el modo de mantener los conceptos del dominio. Se ha abandonado la *jerarquía conceptual* y hemos pasado a una representación más académica (descrita en la sección 6.5) que nos proporciona beneficios adicionales.

Como se decía en el capítulo 4, tras la elección de los conceptos es necesario conseguir un ejercicio que los ponga en práctica. Esta fase se puede realizar mediante la *creación al vuelo* del ejercicio, o mediante *su recuperación*. Como hicimos en la primera instanciación, mantenemos la idea de que resulta más interesante la aproximación *basada en casos* al resultar mucho más fácil el proceso de *extracción del conocimiento* de los expertos.

No obstante, ya hemos dicho que el modo de guardar los ejercicios en la primera versión de JV^{2}M tenía deficiencias por el laborioso proceso de autoría que necesitaba. En esta reencarnación, sin embargo, nos beneficiamos del modo en el que se almacenan los conceptos del dominio para desplazarnos hacia una aproximación *rica en conocimiento*. Esto alivia el tiempo necesario para la autoría (especialmente una vez que la barrera de los primeros ejercicios ha sido superada) gracias a la *reutilización* del conocimiento general del dominio entre diferentes ejercicios.

Siguiendo nuestra filosofía de desarrollo de videojuegos educativos, la resolución del ejercicio no podría realizarse de otro modo que no fuera integrada en un nivel del juego. Así, las mecánicas de la parte lúdica quedan entretejidas con los pasos de resolución de manera que resultan inseparables. El modo en el que esto se ha realizado quedó descrito en la sección anterior dedicada al *diseño* del videojuego.

Para poder considerarlo un *sistema educativo*, es necesario que sea capaz de seguir los pasos al jugador para saber si está resolviendo o no correctamente el ejercicio. En nuestro caso, esta labor se realiza, como no podía ser de otro modo, a lo largo de *todo* el proceso de resolución del ejercicio.

Naturalmente, para poder hacerlo es necesario que el programa *disponga de la solución* del ejercicio que el estudiante está resolviendo. En la primera instanciación de JV^{2}M la solución venía inseparablemente unida al propio ejercicio recuperado. Sin embargo, en el capítulo 4 comentamos que cuando los ejercicios se *recuperaban* de una base de casos se disponía, en realidad, de dos aproximaciones. La primera es, como hemos dicho, mantener la solución junto con el propio enunciado dentro de la batería de ejercicios disponibles. La segunda es disponer de un *resolutor* independiente de los ejercicios del

dominio. Esto permitiría incluso que el propio alumno propusiera al sistema ejercicios nuevos y que éste los resolviera a modo de ejemplo.

En el caso de la primera versión de JV^{2}M, la necesidad de incluir la solución junto con cada ejercicio venía impuesta por el modo de proporcionar ayuda al estudiante. Esto además incrementaba drásticamente el trabajo de autoría de los ejercicios, que debían ser intensamente *anotados* por los expertos para añadir las explicaciones.

En esta nueva instanciación hemos seguido un modelo diferente. Aprovechándonos de la nueva forma de mantener los conceptos del dominio, disponemos de un *resolutor genérico basado en casos* (sección 6.8). Esto exige un nuevo frente de trabajo, pues hay que dotar al sistema de la capacidad para resolver los ejercicios *y explicar* sus soluciones. Pero a la vez aligera aún más la autoría *de ejercicios*.

Para proporcionar información al usuario hemos perdido la posibilidad de mantener un diálogo completamente controlado por el ordenador, dado que los textos "enlatados" que estaban disponibles en JV^{2}M 1 han desaparecido con el nuevo modelo de conocimiento.

El sistema ahora es capaz de recibir e interpretar *texto libre* por parte del usuario. La consulta es analizada y se detecta si tiene relación con los aspectos *de compilación* del ejercicio o con la JVM o el entorno virtual. En el primer caso se aprovecha la transparencia del *resolutor* de los ejercicios para obtener explicaciones, mientras que en el segundo se hace uso de CBR textual para analizar la documentación escrita que el sistema posee y proporcionarle aquella que más parezca adaptarse a la pregunta (sección 6.9).

La figura 5.3 muestra el esquema con el funcionamiento a alto nivel de la aplicación. Como ocurría en el caso de JV^{2}M 1, es una particularización del modelo general del capítulo 4. La primera etapa sigue realizándose sin hacer uso de ninguna técnica especial de Inteligencia Artificial, y los ejercicios se extraen de una base de casos. Ahora, sin embargo, el ejercicio recuperado *no* trae la solución, de manera que en la etapa de comprobación de las acciones del usuario es necesario *resolver* automáticamente el ejercicio al que se le ha enfrentado. Para eso se utiliza también un sistema CBR. Esta etapa proporciona directamente la información a suministrar al estudiante, aunque ahora no se le interrumpe salvo que lo haya pedido explícitamente.

En lo referente al componente de videojuego, el sistema tímidamente adapta la dificultad de la parte lúdica. Para eso, utiliza una serie de reglas sencillas que le permiten ajustar el número de "portarecursos" disponibles en función de la habilidad que el estudiante haya mostrado previamente. Con esa información, el nivel de juego es construido de forma procedimental para crear los diferentes "pisos" y colocar en ellos los portarecursos asociados a cada uno de los métodos Java.

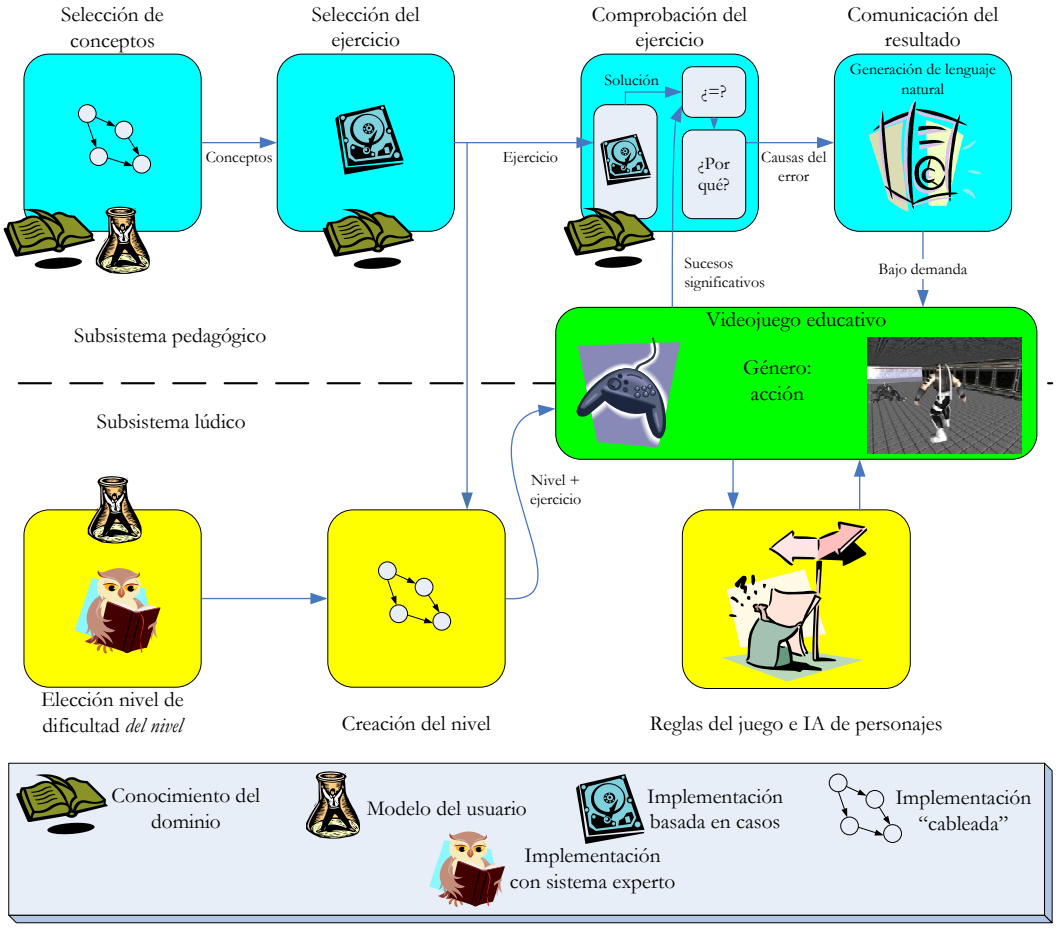


Figura 6.4: Esquema lógico del funcionamiento de JV^{2}M 2

6.5. La ontología

Nuestro modelo de aplicaciones educativas, con su división de la ejecución en las cinco etapas, exige un modo de que la primera fase le proporcione a la segunda la lista de *conceptos* a poner en práctica en el siguiente ejercicio. Para poder hacerlo, es obvio que necesitamos mantener de algún modo en el sistema la *lista de conceptos* del dominio que enseñamos.

Este modo de proceder también hace intuitivo el uso de modelos del usuario basado en plantillas, marcando qué conceptos de dicha lista sabe ya el estudiante, y cuales no. En la primera versión de JV^{2}M usábamos la *jerarquía conceptual* (sección 5.6) como *contenedor* de los conceptos del dominio, que eran referenciados tanto en el modelo del estudiante como en el módulo pedagógico para comunicarse con la segunda etapa.

Sin embargo, la jerarquía conceptual también la utilizábamos como almacén de *explicaciones* en lenguaje natural que eran proporcionadas ("leídas") al alumno en los momentos oportunos. Además, las relaciones entre los diferentes conceptos también quedaban anotadas con *preguntas* que servían para guiar la conversación mantenida entre el usuario y el sistema, de un modo parecido al encontrado en los juegos de aventura.

Este modo de proceder muestra que la jerarquía conceptual se creó pensando en la fase de comunicación con el usuario, pero no es posible utilizar las relaciones entre conceptos internamente para nada más porque no poseen "semántica". Por ejemplo, dentro del grupo de los conceptos relativos a las estructuras de la JVM tenemos el concepto "Pila de operandos" (`operandStack`), que se relaciona con el concepto "Pila" (`stack`) a través de la arista etiquetada con "¿Qué es una pila exactamente?". Este concepto a su vez está enlazado con "Uso de la pila de operandos" (`usingOperandStack`) a través de la pregunta "Vale, ya sé qué es una pila. Pero ¿cómo se usan para los operandos?". Desde el punto de vista del sistema, ambas conexiones *son iguales*. Sin embargo, desde el punto de vista conceptual, la relación entre `operandStack` y `stack` es una relación "es un", mientras que la que une `stack` y `usingOperandStack` es completamente artificial para que la conversación y las opciones disponibles queden realistas.

El ejemplo pone de manifiesto que la jerarquía conceptual de JV^{2}M 1 *no* es una formalización, sino más bien un *mapa conceptual* construido teniendo en mente la fase de comunicación con el usuario al guiar las aristas con preguntas.

No obstante, como la jerarquía se utiliza para otras partes del sistema, los conceptos están marcados indicando a qué categoría de las cinco disponibles pertenecen. Esto es especialmente importante para el uso que de la jerarquía conceptual realizan el modelo del usuario y el módulo pedagógico. Como se dijo, estos dos componentes sólo utilizaban las dos primeras categorías (conceptos de compilación e instrucciones de la JVM) para anotar lo que el estudiante sabe, o para pedir ejercicios de un determinado tipo.

Por otro lado, los grafos de ejecución necesitan información sobre el "prototipo" de las instrucciones y las microinstrucciones, para que la herramienta con la que se construyen pueda garantizar la coherencia. Para cubrir esta necesidad, de nuevo enriquecíamos la jerarquía conceptual con datos adicionales para esas dos categorías de conceptos. La conclusión de todo esto es que la jerarquía conceptual de JV^{2}M 1 es un punto central de todo el sistema, que aglutina información de muy diversa índole para poder dar servicio a sus diferentes componentes.

Lo que estamos buscando es mantener el conocimiento del dominio de una manera más estructurada, añadiendo "semántica" que el *sistema* pueda utilizar para *razonar*, y no tan sólo textos que el estudiante comprenda. La Inteligencia Artificial ha lidiado con el problema de la *representación del co-*

nocimiento desde sus inicios, por lo que recurrimos a sus últimas técnicas para aprovechar las diferentes tecnologías que ha ido generando a lo largo de los años. Gracias a esto esperamos conseguir una representación válida *internamente* para el sistema, pero, por desgracia, a la vez nos estaremos alejando del "texto enlatado" que teníamos en la jerarquía conceptual y que tan cómodo resultaba para comunicarnos con el usuario. Por tanto, esta representación no podremos utilizarla directamente para proporcionar información al estudiante, sino que necesitaremos algún procesado posterior que nos salve esta distancia.

De las diferentes opciones disponibles, hemos utilizado como formalismo para representar el conocimiento las *ontologías*. En filosofía, *ontología* es la "ciencia del ser" que se dedica a su definición, y a la categorización de todo lo que existe. En informática se refiere a un exhaustivo y riguroso esquema conceptual que modeliza un dominio dado, y que puede utilizarse para compartir información entre varios sistemas o componentes. Para el filósofo, por tanto, la *ontología* intenta responder a la pregunta "¿qué tipo de cosas existen?", mientras que los informáticos acotan este objetivo a "¿qué tipo de cosas *deberían ser capturadas y representadas?*" (Sharman et al., 2007, página XI).

Para poder trabajar con ontologías es necesario un modo de *representarlas*, es decir disponer de un *lenguaje de ontologías* para codificarlas. La Unión Europea desarrolló OIL (*Ontology Inference Layer*, Capa de Inferencia con Ontologías) como su propuesta para representar ontologías, basada en sintaxis RDF(S). Simultáneamente, en Estados Unidos el Departamento de Defensa a través de DARPA (*Defense Advanced Research Projects Agency*, Agencia de Proyectos de Investigación Avanzados de Defensa) planteó su propia alternativa conocida como DAML (*DARPA Agent Markup Language*, Lenguaje de Marcas de Agentes de DARPA), también basada en RDF.

En Marzo de 2.001 las dos iniciativas se fusionaron en un intento de crear un lenguaje de comunicación entre agentes de ámbito mundial, al que se le bautizó (sin demasiado esfuerzo) como DAML+OIL.

En él se basó el W3C (*World Wide Web Consortium*, Consorcio de la WWW) para añadir una capa de soporte ontológico sobre XML y RDF(S) con la intención de añadir la posibilidad de realizar definiciones ricas en semántica con ella. El resultado fue OWL (*Web Ontology Language*, Lenguaje de Ontologías para la Web), que se ha destacado como la alternativa más atrayente para especificar ontologías (Lacy, 2005).

Existen tres "dialectos" (o "especies"[3]) de OWL. El primero de ellos se conoce como OWL Full, y constituye el *lenguaje completo* en su máximo ex-

[3]Aquí aparece un giro humorístico al llamar "especies" a las diferentes versiones del lenguaje, dado que OWL es "lechuza" (o búho) en inglés. De hecho, el acrónimo correcto de *Web Ontology Language* debería ser WOL y no OWL. Sin embargo, se escogió OWL para conseguir un nombre fácil de pronunciar (en inglés) y del que fuera obvio construir un logotipo. Además, normalmente las lechuzas y búhos son asociados con la sabiduría.

plendor, proporcionando toda la expresividad posible sin restricciones sobre RDF. El problema de OWL Full es que la expresividad está reñida con la posibilidad de razonar con ella en tiempo finito.

Para solucionar esto, el segundo dialecto, conocido como OWL DL restringe el modo en el que algunas construcciones pueden ser utilizadas. Esta pérdida de expresividad proporciona un lenguaje de descripción que, como contrapartida, permite un razonamiento eficiente con las llamadas *lógicas descriptivas*. OWL DL consigue así un compromiso entre la expresividad y el razonamiento decidible (es decir, aquél que termina en un tiempo finito).

El último y más sencillo dialecto es el conocido como OWL Lite, que restringe todavía más las posibilidades expresivas. Está enfocado a aquellos usuarios que desean beneficiarse del uso de ontologías sin tener que dedicar excesivo tiempo en codificar relaciones semánticas complejas. De esa forma, diferentes organizaciones pueden actualizar sus orígenes de datos en XML y RDF a OWL Lite, sin tener que sofisticar en exceso sus herramientas, pues las capacidades de razonamiento que se soportan con la semántica de OWL Lite resultan relativamente sencillas de implementar.

La principal ventaja del uso de OWL estriba en que existe un gran impulso de la comunidad al ser el estandarte de la conocida como *Web Semántica*. De ese modo, hay disponibles diferentes herramientas para la *creación* de ontologías que utilizan OWL como modo de representarlas (por ejemplo Protégé[4]) y existen librerías de programación para leerlas y manipularlas (como Jena[5]). Además, hay disponibles diferentes *razonadores* que son capaces de recibir las ontologías para extraer de ellas conocimiento adicional (si se utiliza OWL DL).

Una vez que decidimos utilizar ontologías hay que preguntarse, como decíamos, qué tipo de cosas del dominio deberían ser capturadas y representadas. Si analizamos la jerarquía conceptual de JV^{2}M , de las cinco "categorías" de conceptos que teníamos, las dos inferiores (estructuras de la JVM y metáfora) sólo se utilizaban para la comunicación con el usuario durante las conversaciones. Hemos dicho que la nueva representación no la queremos utilizar para este cometido, sino como modelo interno de conocimiento para razonar. Por tanto, no nos preocuparemos de formalizar el conocimiento de estas dos categorías en nuestra ontología.

De la misma forma, en JV^{2}M 2 la categoría de las microinstrucciones también ha perdido peso. Originalmente sus conceptos se utilizaban tanto en la parte de comunicación con el estudiante como para referenciarlas desde las aristas de los grafos de ejecución. Sin embargo, en el nuevo diseño hemos dejado de forzar al estudiante a ejecutar manualmente las instrucciones de la JVM, por lo que los grafos de ejecución dejan de tener sentido (al menos en lo que se refiere a necesidad de conocimiento explícito), y con ellos también

[4] `http://protege.stanford.edu/`
[5] `http://jena.sourceforge.net/`

la categoría de las microinstrucciones en la jerarquía conceptual.

Esto nos deja únicamente las dos categorías superiores, relativas a los conceptos de compilación y a las instrucciones (que no microinstrucciones) de la JVM. Por comodidad, hemos creado "dos" ontologías diferentes especificadas con OWL en dos ficheros distintos, utilizando la herramienta Protégé. La primera de ellas modeliza el lenguaje Java. Para ello, utilizamos la riqueza de especificación que proporciona OWL DL. De esta forma, no solamente nos preocupamos de crear conceptos que tengan interés durante la comunicación con el usuario, sino que podemos tener conceptos genéricos que permitan categorizar otros de manera automática. Por ejemplo, el concepto `NamedElement` define todos los elementos de un programa Java que tengan nombre (como variables, métodos, clases o paquetes).

Además, las relaciones entre los conceptos pueden ser más complejas y disponer de *semántica*, algo que en la jerarquía conceptual de JV^{2}M 1 echábamos en falta. Por ejemplo, tenemos los conceptos `Variable` y `Type`, y hacemos explícita la idea de que cualquier `Variable` debe tener un `Type`. Además, creamos individuos (instancias) del concepto `Type` con los diferentes tipos primitivos de Java (`int`, `long`, etcétera). Lo interesante es que la riqueza de la representación ontológica nos permite especificar mucho más conocimiento que *el sistema podrá aprovechar* más adelante.

Por otro lado, tenemos la segunda ontología que se refiere a las instrucciones de la máquina virtual. Mantenemos así información sobre relaciones entre instrucciones creando conceptos generales (por ejemplo, `bipush` es una `stackInstruction`). En realidad en la jerarquía conceptual de JV^{2}M manteníamos bastante información sobre ellas, pues guardábamos su "prototipo". Todo ese conocimiento se mantiene también en la ontología en OWL, pero se enriquece con información concreta sobre los tipos de los operandos incrustados en el propio "opcode"[6] de la instrucción, o los esperados en la pila de operandos del *frame* actual.

Aunque estén especificadas en dos ficheros diferentes, en realidad entre ellas existe una estrecha relación y hacen referencia la una a la otra. Por ejemplo, la información sobre los tipos de los operandos de las instrucciones de la JVM se especifica utilizando los individuos del concepto `Type` de la ontología sobre Java.

Como ya hemos dicho, el primer uso de las ontologías se realiza durante la toma de la decisión de qué conceptos debería practicar el estudiante en el siguiente episodio de aprendizaje. Para eso, seguimos utilizando un modelo de usuario basado en plantillas, y el conocimiento del currículo consiste en la secuencia de conceptos que los expertos del dominio han considerado más adecuada. La diferencia respecto a la primera versión de JV^{2}M es que ahora los conceptos se especifican dentro de una ontología, en lugar de estar

[6] *Operation code*, código de operación.

definidos de una manera "ad hoc" en la jerarquía conceptual. En principio esta diferencia puede resultar bastante nimia. Lo interesante es que la ontología proporciona mucha más semántica que podremos utilizar en el resto de etapas de la aplicación tal y como iremos viendo en el resto de secciones.

6.6. Los ejercicios

En JV^{2}M 1 los ejercicios se mantenían en una base de ejercicios creada de antemano, en lugar de ser construidos al vuelo. Para crearlos se utilizaba una herramienta específica a través de la que se anotaba tanto el enunciado como su solución con explicaciones sobre por qué el código se había compilado del modo en el que se había hecho.

Aunque esto proporcionaba al sistema un modo cómodo de comunicarse con el estudiante a través del diálogo, el trabajo de crear un ejercicio (incluso sencillo) era muy laborioso y propenso a fallos. La labor se asemejaba a construir los árboles de diálogo de los juegos de aventura, con los que hay que tener un gran cuidado para mantener la coherencia.

En JV^{2}M 2 seguimos haciendo uso de una base de ejercicios, pero ahora utilizamos una aproximación mucho más cómoda, dado que éstos no tienen que guardar ni su solución ni, por tanto, las explicaciones que indiquen por qué se ha compilado el código de esa manera. Naturalmente, la comodidad que ganamos aquí supondrá un esfuerzo extra en las etapas de análisis de la solución y de comunicación con el estudiante que veremos más adelante.

La primera aproximación para mantener los ejercicios es relativamente sencilla. La etapa anterior de selección de conceptos nos ha dado aquellos conceptos de la ontología que el estudiante debe practicar. Cada ejercicio está compuesto por uno o varios ficheros de código fuente (`.java`), que están indexados precisamente mediante los conceptos de la ontología.

Hay varias formas de mantener esa información. Por ejemplo, no resultaría demasiado complicado mantener en un fichero XML una lista con la información de cada ejercicio (nombres de sus ficheros `.java`) y los nombres de los conceptos de la ontología que pone en práctica. En ese caso, realizaríamos una recuperación *computacional*, recorriendo de manera manual los datos mantenidos en el XML buscando un ejercicio adecuado.

Esta alternativa, sin embargo, no aprovecha en exceso la batería de herramientas de apoyo disponibles alrededor de OWL. En lugar de reinventar la rueda, utilizamos también la notación ontológica como mantenedor de nuestra base de casos. En concreto, en un fichero OWL nuevo con la información de *todos* los ejercicios crearemos un *individuo* que sea instancia de cada uno de los conceptos que pone en práctica cada ejercicio. Esos individuos están anotados con el nombre del ejercicio que, externamente al fichero OWL, estará relacionado con los ficheros de código fuente.

```
class Arithmetic1 {
        public static void main (String [] args ) {
                int a, b, c;
                a = 7;
                b = a;
                c = 3 + 1;
        }
} // main
```

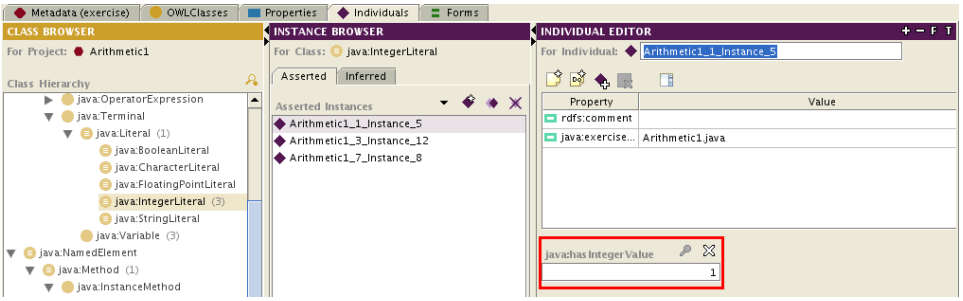


Figura 6.5: Ejemplo sencillo y su correspondiente OWL para recuperación

Por ejemplo, en la figura 6.5 se muestra el código fuente de un sencillo ejemplo, y algunos de los individuos que habría que añadir a la ontología de ejercicios para representarlo. Puede verse que hay tres individuos que son instancias del concepto `IntegerLiteral`, uno por cada constante entera que aparece en el código. Vemos incluso que el individuo seleccionado (`Arithmetic1_1_instance_5`) tiene una propiedad (`hasIntegerValue`) que especifica el valor entero que posee. También tiene una anotación con el nombre del ejercicio. Son además visibles tres individuos `Variable` (que hacen referencia a las tres variables del programa), y un `Method`.

Esta idea se repite con todos los ejercicios disponibles, añadiendo los conceptos correspondientes a todos ellos. Cuando el modelo pedagógico nos proporciona a partir del perfil del estudiante aquellos conceptos que se deben practicar, recuperamos aquellos individuos que sean instancias de los conceptos solicitados. Esto supone un desplazamiento hacia una recuperación *representacional* (Porter, 1989). De hecho, apoyándonos en la información de la ontología el módulo pedagógico podría solicitar practicar conceptos de "orden superior" que no tuvieran individuos directamente, y la recuperación proporcionaría individuos de sus subconceptos.

La elección del siguiente ejercicio se sofistica aún más si pensamos que el modelo pedagógico nos puede proporcionar información sobre conceptos que *no* deben ser puestos en práctica. Desde el punto de vista pedagógico, en un

determinado momento se asocian al alumno unos conocimientos adquiridos, y en cada episodio de aprendizaje se quieren poner a prueba conceptos nuevos. Es necesario que el ejercicio recuperado incluya *esos conceptos* nuevos, y no resulta dañino que contenga otros previamente conocidos (su inclusión en el ejercicio es, en cierto modo, "opcional"), pero definitivamente *no* querremos que incluya el resto de conceptos no conocidos y que no se quieren enseñar aún. Con esta información, la comunicación entre la primera y la segunda fase se enriquece, y durante la elección del siguiente ejercicio tendremos que *realizar un filtrado* para eliminar los ejercicios que requieren conocimientos que están aún más allá de las posibilidades de nuestro alumno.

En cualquier caso, a partir de todo lo anterior podemos resumir que la labor de autoría de ejercicios queda ahora reducida a escribir el código Java, y a crear un fichero OWL que importe la ontología de conceptos sobre Java para crear individuos de aquellos que el ejercicio pone en práctica. Esta labor es mucho más llevadera que la necesaria en la primera versión de JV^{2}M, pero aún puede mejorarse más. Al fin y al cabo, los conceptos que un ejercicio practica están implícitos en su enunciado, y éste está lo suficientemente estructurado (es código fuente, después de todo) como para que se pueda realizar un análisis automático para extraerlos.

Con este objetivo se ha desarrollado la herramienta `java2owl` que analiza una batería de ficheros Java y escribe un fichero OWL tal y como lo espera posteriormente el sistema. Por comodidad, en lugar de realizar manualmente el análisis sintáctico del código Java se ha utilizado Java2XML[7], una herramienta que convierte un fichero Java a XML siguiendo la DTD JavaML[8]. Esto convierte el laborioso análisis sintáctico en la lectura de un fichero XML que puede apoyarse en los diferentes analizadores XML disponibles. Para generar el fichero OWL, se utiliza la librería Jena que esconde los detalles de la sintaxis RDF(S) para generar ficheros OWL válidos.

Con esta herramienta, la labor de autoría se reduce todavía más, pues es suficiente con disponer del código Java de programas sencillos y alimentar con ellos a la herramienta. Comparándolo con la situación en JV^{2}M 1 nos encontramos en una situación mucho más positiva, dado que resulta mucho más barato disponer de una base de ejercicios bien poblada. En cualquier caso, comparar ambas alternativas es aún un poco arriesgado, dado que con la aproximación seguida aquí *no* disponemos de la solución de los ejercicios recuperados, ni tenemos la capacidad de suministrar ayuda y explicaciones al estudiante. Será necesario esperar a detallar las soluciones adoptadas para cubrir estas carencias antes de poder tomar una decisión informada sobre cual de las dos resulta menos exigente en tiempo.

Una sofisticación adicional que resulta posible al disponer de la ontología

[7] `https://java2xml.dev.java.net/`
[8] `http://javaml.sourceforge.net`

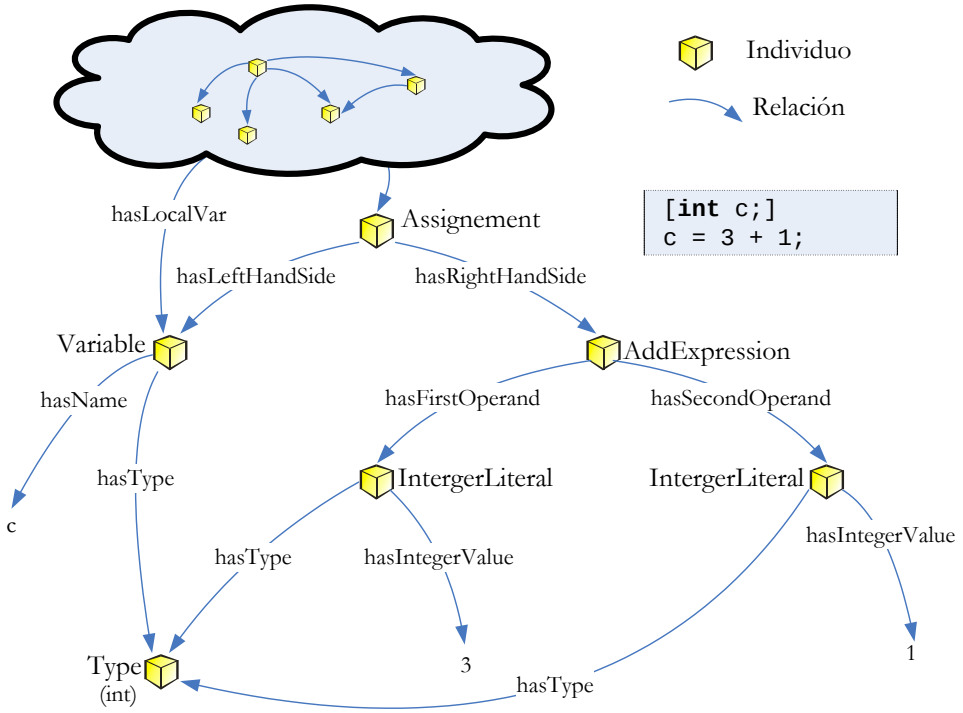


Figura 6.6: Fragmento de un ejercicio codificado en OWL

es mantener el programa Java no como código fuente, sino como un grafo de individuos de la ontología relacionados con las diferentes propiedades disponibles. Gracias a esto tenemos una descripción estructurada en OWL (con *individuos*) que es clasificada e indexada a través de los conceptos de la ontología (Gómez Martín et al., 2005c).

Por ejemplo, para la sencilla expresión:

$$c = 3 + 1$$

utilizaríamos la representación de la figura 6.6. En ella se muestran los *individuos*, aunque aparezcan etiquetados con el *concepto* del que son instancia. El código Java tiene una variable local (individuo del concepto `Variable`) que se utiliza como lado izquierdo (`hasLeftHandSide`) en una asignación (`Assignement`). Ésta tiene como parte derecha una expresión de suma (`AddExpression`) cuyos dos operandos son literales enteros (el `3` y el `1`).

Es necesario destacar que los individuos que aparecían en la figura 6.5 se utilizaban *para recuperación*, por lo que formaban parte de la *descripción* del caso. Los que mostramos ahora, sin embargo, especifican el *ejercicio*, por lo que desde el punto de vista del sistema CBR forman la *solución* (es decir

lo *recuperado* de la base de casos).

Tener el código Java estructurado de esta manera nos permite realizar cosas más sofisticadas de las que podríamos realizar manualmente sobre el código en texto plano. Por ejemplo, a partir de la representación de la figura anterior resulta fácil deducir *de manera automática* que el tipo de la expresión de suma es también de tipo entero sin tener que construir un analizador del lenguaje Java. El precio que hay que pagar es, naturalmente, que conseguir una representación ontológica no es trivial. Como exigir una construcción manual por parte de expertos sería una vuelta a una autoría increíblemente costosa y propensa a fallos, la solución es mejorar la herramienta `java2owl`. Su implementación es algo tediosa, pero desde luego posible, dado que es un trabajo totalmente especificado y repetitivo.

Gracias a la *estructura* con la que dotamos a nuestros ejercicios conseguiremos ventajas en el resto de etapas que quedarán claras en las próximas secciones. Pero además permite añadir *adaptación* en la fase de recuperación de ejercicios de la base de casos. Para explicarlo, revisemos los componentes de los que está formada una *consulta* a este sistema CBR:

- Un conjunto de conceptos C_i que el usuario tiene que practicar en el siguiente ejercicio.
- Un conjunto de conceptos L_j con los conceptos que el estudiante *ya conoce*. No se exige que el ejercicio recuperado incluya estos conceptos, pero si aparecen no resultan perjudiciales.
- Un conjunto de conceptos N_k que *no deberían* estar incluidos en el ejercicio. El estudiante aún no los conoce, y el módulo pedagógico ha decidido que aún es demasiado pronto para que se practiquen. En realidad, puede obtenerse eliminando de la lista de *todos* los conceptos del dominio representados en la ontología aquellos que aparecen en los dos conjuntos anteriores.

A partir de una consulta q con esa información, la *recuperación* tiene dos etapas:

1. Filtrado:
   - Se obtiene el conjunto de todos los ejercicios que incluyan a los conceptos que hay que practicar. A ese conjunto le denominaremos R.
   - Si R es vacío, entonces se busca al individuo (ejercicio) más específico, es decir aquél que incluya una mayor cantidad de conceptos C_i.
2. Selección:

```
public class Example {
    public static void main (String params) {
        int a;
        a = 12;
        if (a < 5) {
            int c;
            c = a + 3;
        }
    }
}
```

Figura 6.7: Ejercicio recuperado

- Se analizan los diferentes ejercicios de R para devolver el más similar a la consulta teniendo en cuenta el resto de componentes de la consulta (L_j y N_k).

Si el filtrado es afortunado, podríamos encontrar varios ejercicios que pongan en práctica los conceptos C_i, pero aun así tener mala suerte y arrastrar también conceptos prohibidos N_k.

Por ejemplo, asumamos que el módulo pedagógico ha solicitado un ejercicio que cumpla las siguientes condiciones:

- C_1 = `MultiplicativeExpression`
- L_1 = `Variable`, L_2 = `Assignment`
- N_1 = `If`, N_2 = `While`

En realidad, la lista de conceptos N_k sería mucho más larga, pero nos sirve como ejemplo. En este contexto, ante una base de casos poco poblada la fase de *recuperación* del ciclo CBR podría proporcionarnos el ejercicio de la figura 6.7. Es significativo observar que el concepto que queríamos practicar involucraba una expresión multiplicativa, y sin embargo el ejercicio no dispone de ninguna. En su lugar, la recuperación ha encontrado una expresión de suma. Dado que ambos conceptos están cercanos en la ontología (son hermanos) la selección ha determinado que la similitud entre ambos era suficiente como para considerarse un ejercicio razonablemente válido. Por desgracia, el ejercicio incluye también la construcción `If`, que estaba explícitamente prohibida en la consulta.

Ante este panorama, el sistema CBR que se encarga de proporcionar el ejercicio debe realizar *adaptación*, es decir modificar el caso recuperado para ajustarlo a la consulta recibida. Realizar los cambios exigidos por la adaptación directamente sobre el código fuente es posible, pero resulta bastante tedioso de programar y, por lo tanto, también muy propenso a errores.

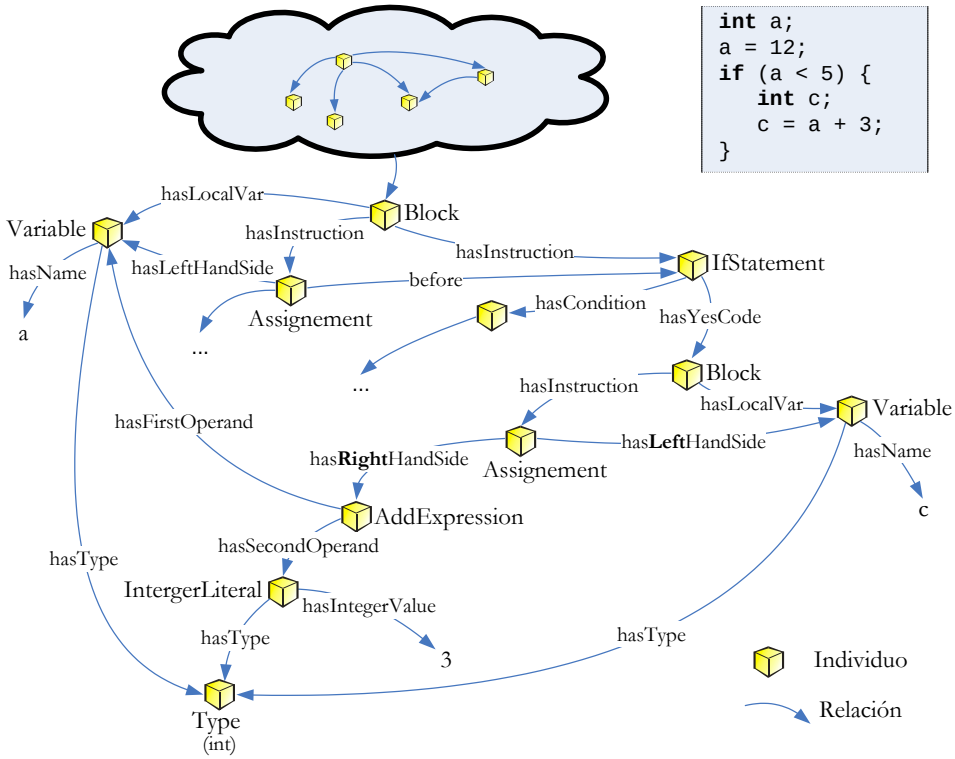


Figura 6.8: Fragmento del grafo de individuos del ejercicio de la figura 6.7

Afortunadamente, el sistema no recuperaría el código fuente, sino su versión estructurada en forma de individuos OWL. En la figura 6.8 aparece un fragmento del grafo de dichos individuos.

La ventaja de esta representación es que el esquema de adaptación fue formalizado por González Calero et al. (1999) en función de eliminaciones y sustituciones. Por un lado se aprovecha del conocimiento de similitud que puede ser extraído de la ontología. En concreto, como ya hemos visto, la *cercanía* entre dos conceptos indica su parecido, dando información útil sobre qué individuos podrían ser *sustituidos* por otros sin peligro.

Por otro lado, las dependencias entre los distintos elementos de la solución se representan *explícitamente* mediante relaciones entre individuos. La ontología se puede enriquecer con un conjunto de *reglas* que hagan explícito el tipo de *cambios* que podemos realizar en unos individuos y cómo afectan a aquellos con los que están relacionados. Dichas reglas son utilizadas por el sistema para realizar los cambios adecuados.

En función del modo en el que se almacenen dichas "reglas", estaremos en el caso de una representación procedimental (si las *programamos*), o una representación declarativa (si las escribimos para que un "motor" las inter-

prete y lleve a cabo). Nosotros hacemos uso de jCOLIBRI (Bello Tomás et al., 2004), un armazón para el desarrollo de sistemas CBR escrito en Java. jCOLIBRI es un *framework* modular, existiendo diferentes ampliaciones particularizadas al tipo de sistema CBR para el que se vaya a utilizar y que, en concreto, dispone de soporte para ontologías, lógicas descriptivas y reglas para la adaptación en un formato propio.

De manera general, la adaptación realizada propaga los cambios existentes entre la descripción solicitada por el usuario y la descripción del caso recuperado de la siguiente manera:

1. Se obtiene la lista de individuos que deben ser *adaptados*, A. Esta lista está formada por dos tipos de elementos:

   a) Aquellos que dependen de una característica de la descripción del caso que *no* aparece de ningún modo en la consulta, y tendrán que ser eliminados (A_E).

   b) Aquellos que deben ser sustituidos por un valor diferente (pero cercano) que sí aparece en la consulta (A_S).

2. Cada elemento de L es eliminado o sustituido por un nuevo elemento. Para eso, se vigilan las dependencias de los individuos antiguos para no dejar en un estado inconsistente el resultado. La estructura del grafo y la semántica de las relaciones entre los individuos facilitan esta labor.

En el caso recuperado en el ejemplo, tenemos por un lado un elemento que debe ser eliminado ($A_E = \{\texttt{If}\}$) y por otro un elemento que debe ser sustituido ($A_S = \{\texttt{MultiplicativeExpression}\}$). La adaptación elimina el individuo `If`, pero teniendo en cuenta su dependencia con el bloque de código asociado al cumplimiento de su condición. Para eso, la relación la "sube" hacia el bloque en el que él está contenido. Por su parte, las restricciones exigidas en el concepto `AddExpression` y en el que se utilizará para sustituirle (`MultiplicativeExpression`) son las mismas, por lo que la sustitución no ocasiona ningún efecto lateral alrededor.

Tras la adaptación, el grafo (también parcial) de individuos que resulta es el mostrado en la figura 6.9. En él podemos observar que la instancia del `if` ha desaparecido, y la suma es ahora una multiplicación.

Obviamente, este grafo de instancias OWL no es interesante para el estudiante. Él espera que se le proporcione el enunciado del ejercicio como código fuente escrito en Java. Antes de disponer de adaptación, podíamos mantener de hecho las dos representaciones (estructurada en OWL y código fuente) sin ningún problema, y utilizar una u otra según nos interesara. Sin embargo, como ahora el sistema altera la representación estructurada, necesitamos un modo de reconstruir la versión en código del nuevo grafo de individuos. Para eso necesitamos `owl2java` que es utilizado *internamente* por el sistema

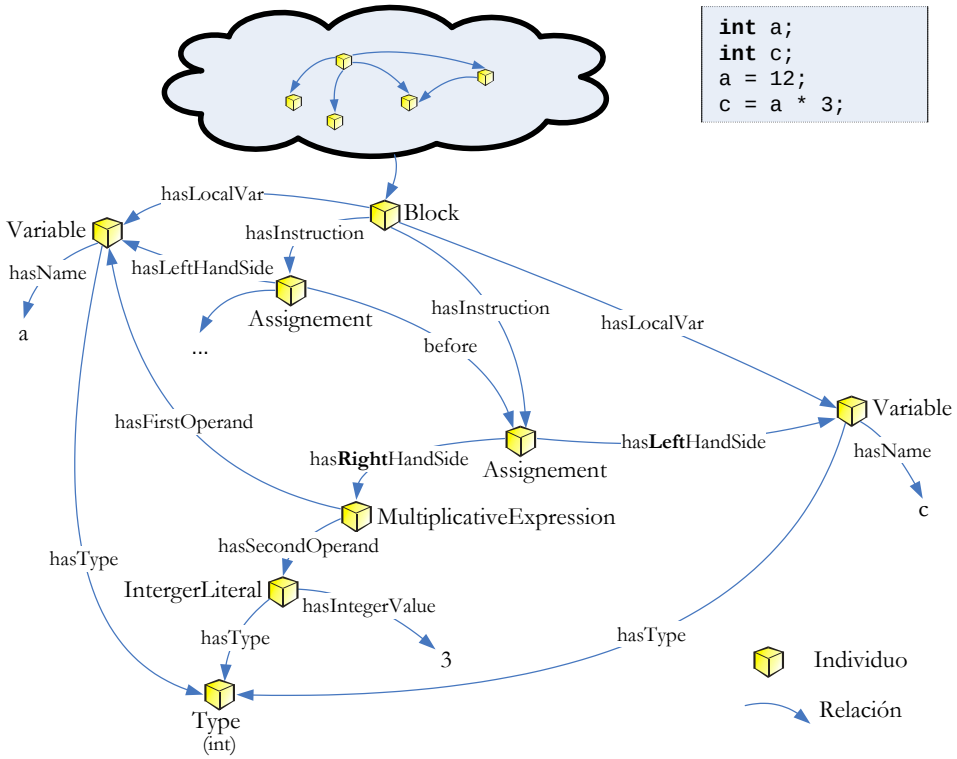


Figura 6.9: Fragmento del grafo de individuos del ejercicio adaptado

cuando lo requiere. Realiza el trabajo inverso al de la herramienta `java2owl` utilizada durante la fase de *autoría*.

El código proporcionado por `owl2java` para el ejercicio adaptado aparece en la figura 6.10. Si se hubiera realizado una adaptación (laboriosa y propensa a errores) basada en el código fuente en texto plano, seguramente *no* se habrían juntado las dos definiciones de variables. Es decir, si se elimina directamente el `if` y su expresión, concatenando los dos bloques de código, se habrían mantenido como las dos primeras líneas la declaración de la variable `a` y la asignación al valor `12`, y luego la declaración de `c` y su asignación. Al utilizar la representación estructurada sólo resulta de interés el orden en las *instrucciones*, no en las declaraciones de variables. De ahí que al aplicar `owl2java` al ejercicio resultante las dos declaraciones aparezcan juntas.

Cuando se habla de manera general del funcionamiento del razonamiento basado en casos, se menciona la etapa de *revisión* de la solución (quizá adaptada) realizada por el sistema CBR. En nuestro caso esa *revisión* (del ejercicio recuperado) pasa por averiguar si el código resultante *es correcto*. Por tanto, a modo de "comprobación de sanidad" de esta etapa, podemos recoger el código devuelto por `owl2java` y proporcionárselo a `javac`. Si éste

```
public class Example {
   public static void main (String params) {
      int a;
      int c;
      a = 12;
      c = a * 3;
   }
}
```

Figura 6.10: Ejercicio adaptado

notifica errores de compilación, claramente habremos sufrido algún problema en la adaptación y el sistema debería ser refinado. En otro caso, podemos estar seguros de que el ejercicio conseguido cumple los requisitos necesarios para ser enseñado al estudiante.

Gracias a la adaptación nos ahorramos todavía más autoría, aunque a costa de necesitar un soporte de herramientas automáticas más exigente. En Díaz Agudo et al. (2005) planteamos un modo de analizar la *cobertura* de una base de casos en un dominio especificado mediante una ontología utilizando FCA. Esta misma idea puede también utilizarse para comprobar los beneficios del uso de adaptación. En cualquier caso, sea o no positiva para ahorrar autoría, la adaptación resulta también cómoda para obtener *variaciones* de ejercicios de manera automática. De hecho, una base de casos podría contener un ejercicio que encajase perfectamente con la consulta solicitada pero *que ya se haya resuelto*. Para evitar la repetición, se podría modificar ligeramente el ejercicio con las mismas ideas expuestas aquí, para añadir variedad sin coste adicional. Esto resulta bastante interesante desde el punto de vista de los expertos, dado que resulta relativamente fácil conseguir un abanico de ejercicios *diferentes*, pero es muy poco motivante tener que crear muchos ejercicios *similares*.

Es necesario destacar que los ejercicios resultantes del proceso de adaptación (o incluso los creados por los tutores y añadidos en la base de casos) pueden resultar poco interesantes. Afortunadamente, el dominio que nosotros estamos enseñando (*traducción de código*) no necesita que los programas proporcionados *hagan* algo útil. Tan solo necesitamos que sea código Java correcto que *compile*. Es precisamente la carencia de una restricción más fuerte en la semántica de los programas propuestos lo que permiten realizar adaptación. No obstante, no está claro si el hecho de que los ejercicios sean poco realistas tendrá algún tipo de repercusión pedagógica.

6.7. La resolución del problema

Los ejercicios son resueltos en un entorno virtual manteniendo nuestra idea de aglutinar un ejercicio y un nivel del videojuego. En la sección 6.3 se ha descrito con cierto detalle el modo en el que la parte lúdica está entrelazada con la pedagógica en el diseño de la aplicación.

Como se ha comentado, ya no se ejecutan las instrucciones manualmente, sino que el estudiante se limita a introducirlas en un terminal. Eso elimina la necesidad de los grafos de ejecución que existía en la primera versión de JV^{2}M, además de hacer desaparecer la sombra de la repetición de los mismos pequeños pasos una y otra vez que podían ocasionar hastío en muchos usuarios.

JV^{2}M 1 también sufría el problema de las instrucciones de salto. Al realizarse de manera paralela la *compilación* y la *ejecución*, el contador del programa lo *encarna* el usuario. Esto originaba problemas en aquellos programas en los que aparecían instrucciones de salto (especialmente en bucles), dado que en lugar de realizar una verdadera traducción, el estudiante realizaba una ejecución "desenrollando" dichos bucles.

Inicialmente JV^{2}M 2 también parecería sufrir este problema, dado que la escritura de instrucciones en el terminal ocasiona su *ejecución*. Para evitarlo, podríamos seguir la solución más simple, que pasa por eliminar completamente esa ejecución. Dejamos que el estudiante se limite a escribir en secuencia las instrucciones, pero éstas no son ejecutadas en ningún momento. El juego pasaría, por lo tanto, a ser un editor de código objeto en el que sólo se pueden escribir instrucciones si se tienen los recursos (operandos) necesarios.

Con esta aproximación, sin embargo, perdemos la posibilidad de que el estudiante *vea* en el entorno la ejecución de cada una de sus instrucciones, y pueda comprender los cambios que ocasionan en las estructuras de la máquina virtual que el entorno representa. Esto tiene importancia pedagógica porque no queremos que los usuarios conozcan de antemano el juego de instrucciones de la JVM, por lo que necesitamos proporcionar su aprendizaje integrado dentro del juego.

Dado, por tanto, que *no* podemos, después de todo, olvidarnos definitivamente de la ejecución, lo que necesitamos en realidad es *separarla* de la parte de traducción, de manera que el usuario escriba por un lado el código objeto, y por otro lado *el sistema* se encargue de ejecutarlo. Así, éste *recupera* el control del contador del programa de manera que si el usuario decide introducir una instrucción de salto a un punto anterior, el sistema (y no el usuario) repetirá la ejecución de las instrucciones implicadas sin necesidad de que el estudiante vuelva a introducirlas.

Para conseguir esta separación es preciso analizar en qué punto estaban traducción y ejecución fusionadas. Obviamente, lo están en la *escritura de una instrucción* en el terminal, pues la escritura supone dos cosas: el alumno

la considera válida como instrucción siguiente para el programa, y además la ejecuta. Pero, en principio, el sistema *no* parece "almacenar" esa instrucción en ningún "contenedor de código objeto".

En vez de seguir por ese camino de final incierto, la solución es cambiar la semántica de la escritura de una instrucción. En lugar de significar directamente su ejecución, en principio tan sólo significa que el usuario la anota como siguiente instrucción válida. El sistema, además, *recuerda* todas las instrucciones escritas por el usuario, en lugar de limitarse a ejecutarlas y aplicar sus efectos en el entorno tal y como ocurría en JV^{2}M 1. De hecho, aparte de instrucciones, podría especificar *etiquetas* para marcar puntos en el código donde querrá volver más adelante[9], e incluso *dejar huecos* que se completarán, quizá, en el futuro (por ejemplo, si la parte `else` de un `if` no debe ejecutarse aún).

Aunque, en principio, el estudiante *no* sea el encargado de "encarnar" el contador de programa, y la escritura de instrucciones esté separada de su ejecución, en la práctica ambas cosas podrían *ocurrir a la vez*. Es decir, cuando el usuario añade una nueva instrucción, una vez conseguidos los recursos adecuados, la aplicación *lanzará* automáticamente su ejecución hasta que se encuentre el siguiente "hueco" en el código compilado.

Para comprenderlo, es mejor utilizar un ejemplo. Supongamos se le encarga al estudiante la labor de traducir el algoritmo de Euclides:

```
public static void main(String args[]) {
  int a,b;
  a = 33;
  b = 6;

  while( a != b ) {
    if( a < b )
      b -= a;
    else
      a -= b;
  } // while
}
```

La versión compilada de este código es:

[9] Para el lector despistado al que los años de programación estructurada hayan grabado a fuego la idea de que las etiquetas son para `goto` y que los `goto` son malos, es necesario recordar que aquí estamos hablando de *código objeto*, y en un lenguaje de tan bajo nivel la única opción es utilizar etiquetas e instrucciones de salto.

```
    ; a = 33
    bipush 33
    istore_0
    ; b = 6
    bipush 6
    istore_1
mientras:
    ; if (a != b) goto salir
    iload_0
    iload_1
    if_icmpeq salir
    ; if (a >= b) goto casoNo
    iload_0
    iload_1
    if_icmpge casoNo
    ; CasoSi: b -= a
    iload_1
    iload_0
    isub
    istore_1
    goto mientras
casoNo:
    ; a -= b;
    iload_0
    iload_1
    isub
    istore_0
    goto mientras
salir:
    return
```

El primer paso del estudiante es introducir la instrucción `bipush 33`. Nada más escribirla el sistema *la ejecuta*, y detecta que no hay ninguna instrucción tras ella, y se detiene. El usuario, al ir consiguiendo más recursos, va escribiendo más instrucciones, que continúan ejecutándose automáticamente. Añade la etiqueta `mientras:` al anticipar que tendrá que volver a ese punto más adelante, y termina escribiendo la instrucción de salto (`if_icmpeq salir`). El sistema determina que la condición *no* se cumple, y queda a la espera de más instrucciones.

El estudiante añade las siguientes instrucciones hasta llegar a la instrucción de salto condicional del `if` interno al bucle (`if_icmpge casoNo`). En este caso la condición *sí* es cierta[10], pero la etiqueta no ha sido introducida aún, por lo que la ejecución queda latente. El usuario *añade un hueco* (para el caso "sí" del `if`), que será completado más adelante, y crea la etiqueta `casoNo`. Eso desplaza la ejecución, que pasa a ese punto, saltándose el hueco.

Poco a poco el estudiante continua añadiendo las instrucciones asociadas al caso `else`, que van, simultáneamente ejecutándose. En el momento en el que el usuario añade la instrucción de salto incondicional `goto mientras` que fuerza la repetición del bucle, la ejecución se desata. El sistema *salta* al principio y *ejecuta* todas las instrucciones hasta volver a encontrarse un hueco. De esa forma, va restando a la variable `a` el valor de la variable `b` *sin intervención del usuario*. En el modelo de ejecución de JV^{2}M 1 todas esas repeticiones del bucle (y de la condición `else` del `if`) deberían ser *reejecutadas* por el usuario manualmente.

Cuando `a` recibe el valor 3 tras varias restas sucesivas, la condición del

[10]Hay que darse cuenta de que la condición escrita en Java y la que aparece en la instrucción de la JVM *son inversas*. En efecto, en Java se comprueba que $a < b$ para ejecutar el caso "sí" mientras que en la JVM se comprueba si $a \geq$b para *saltar* al caso "no".

`if` se cumple y no debe aplicarse el salto al caso `else` que codifica la instrucción `if_icmpge casoNo`. Eso lleva la ejecución automática a un hueco (que el usuario añadió en su momento), y se detiene. El alumno tiene ahora la oportunidad de completar este hueco con las instrucciones del caso sí del `if`.

Cuando añade la última con el salto incondicional (`goto mientras`), de nuevo la ejecución se desencadena, y se parará cuando la condición del bucle deje de cumplirse, y por tanto deba aplicarse el salto de `if_icmpeq salir`. En ese punto, el sistema encuentra una etiqueta sin definir, y de nuevo se detiene. El estudiante la coloca, escribe la última instrucción (`return`), el programa finaliza, y el ejercicio se considera resuelto.

6.8. El análisis de la solución

En JV^{2}M 1 el análisis del estado de la resolución consistía en comprobar si el usuario estaba dando los pasos que estaban marcados como válidos en la solución que se mantenía junto con el ejercicio en el caso recuperado en las primeras etapas. Era necesario poner un poco de cuidado porque el estudiante en realidad realizaba acciones primitivas *sobre el entorno*, y era necesario un proceso de traducción para averiguar la microinstrucción asociada a cada acción. Por su parte, el sistema debía seguir la pista de cual era la *instrucción* actual de código objeto, y cual era la siguiente *microinstrucción* de dicha instrucción.

Conseguir este comportamiento no es excesivamente complicado. Además, la decisión de que los ejercicios guarden su propia solución es ventajosa en la parte de *comunicación* con el estudiante. Como dijimos, el autor del ejercicio debe haber enriquecido con lenguaje natural las relaciones entre código fuente y código objeto. Esas descripciones se utilizan durante el diálogo con el estudiante para explicarle aquellos conceptos que no comprenda.

Por desgracia, también muestra algunas reacciones poco óptimas desde el punto de vista pedagógico, tal y como vimos en la sección 6.2. En concreto, el sistema no siempre identifica correctamente la causa de un error del estudiante, dado que *siempre* supone que el alumno está *compilando bien*. Las explicaciones que Javy proporcionaba ante una equivocación eran siempre relacionadas con la *ejecución*, pues el sistema achacaba el fallo a la realización de las microinstrucciones.

En realidad esta limitación viene del hecho de que JV^{2}M 1 sólo conoce un modo de compilar un determinado código Java, y no concibe la posibilidad de que el alumno haga algo diferente a lo marcado por la solución del ejercicio actual. Esto tiene dos desventajas. La primera es que, efectivamente, el sistema *no* sabe proporcionar ayuda ante un fallo *de traducción*. La segunda es que no se permite la posibilidad de *compilaciones alternativas*.

`bipush 5`
`bipush 4`
`iadd`
	`bipush 4`
	`bipush 5`
	`iadd`
		`bipush 9`

Tabla 6.1: Tres posibles compilaciones de la expresión `5 + 4`

Por ejemplo, la traducción de la expresión de suma `5 + 4` puede tener varios modos de ser compilada, tal y como muestra la tabla 6.1. Si contamos que, además, existe una instrucción *sin operandos* (o más bien, con operando implícito) de forma que `bipush 4` puede ser sustituida por `iconst_4` (y lo mismo ocurre con `bipush 5`), las posibilidades se incrementan aún más. Forzar al estudiante a que genere la solución que el sistema ha considerado como la (única) buena ocasionaba bastante frustración, especialmente porque JAVY insistía en dar explicaciones que *no* tenían relación con la compilación[11].

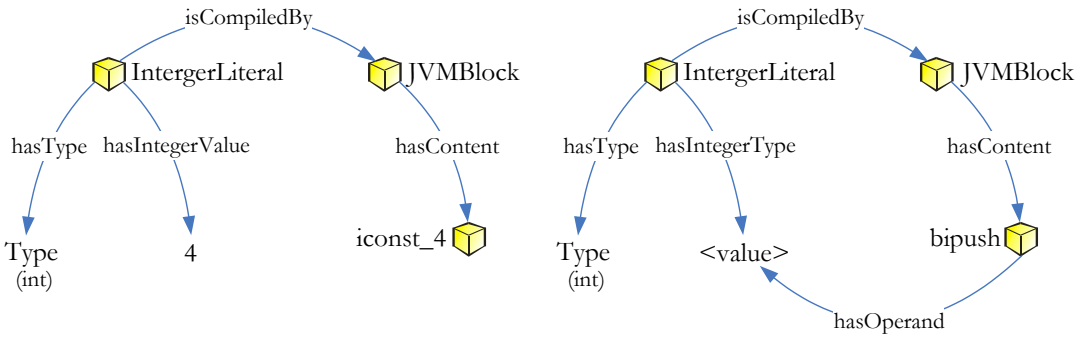
En JV²M 2 la situación es radicalmente diferente. En primer lugar, el estudiante ya *no* ejecuta microinstrucciones, por lo que no hay ambigüedad ante una equivocación: cuando el estudiante comete un error, es por un fallo en la *traducción*. Por otro lado, la solución del ejercicio *no* le acompaña en la base de casos, por lo que no se recupera junto con él. Esto significa que el ahorro de tiempo de autoría del que se hablaba previamente repercute en este punto, en el que recogemos la responsabilidad de *resolver de manera automática* el ejercicio escogido.

Como ya comentábamos en el capítulo anterior, en realidad es muy sencillo disponer de un *sistema experto* que nos solucione *cualquier* ejercicio propuesto: basta utilizar el compilador de Java `javac`. Por desgracia, su uso mantiene el problema de la disponibilidad de una *única compilación*. Ésta, además, estará seguramente bastante optimizada, haciéndola más difícil de comprender. Además, en cualquier caso, el sistema no tendrá modo de *explicar* al estudiante por qué es correcta. Sobreviven así las deficiencias anteriores sobre permitir *varias* traducciones diferentes (pero correctas), y ser capaz de interpretar y explicar compilaciones no válidas.

Para solventar estas deficiencias, el sistema deberá *incluir su propio compilador*. Es decir, necesitamos un "sistema experto" que sea capaz de resolver los ejercicios y que, además, sea "transparente" para el resto de la aplicación. Esta característica resulta vital para poder comprender cómo funciona, por qué toma una determinada decisión en cada punto y si las llevadas a cabo por el estudiante son o no también válidas.

La principal dificultad aquí está en conseguir esa *transparencia*. El área

[11] En realidad la situación no era tan desastrosa, especialmente en un ejemplo tan sencillo. Dado que era la aplicación quien *descodificaba* la siguiente instrucción, el estudiante recibía automáticamente sus operandos (si los había), por lo que podía intuir qué esperaba el sistema de él.



(a) Compilación con `iconst_4` (b) Compilación con `bipush 4`

Figura 6.11: Dos modos de compilar el uso de la constante entera 4

de los compiladores lleva un largo camino recorrido, pero siempre se ha dedicado a construir compiladores que *funcionaran*, no que *explicaran* cómo lo hacían. No nos sirve aquí, por lo tanto, implementar nuestro propia versión simplificada de `javac`, porque seguiría siendo un sistema experto opaco.

La solución es aprovechar que los ejercicios los tenemos representados de manera estructurada en una red de individuos OWL y desarrollar un *compilador basado en casos*. Cada *caso* tendrá como *descripción* una red de individuos OWL representando una *porción* de código Java, y la solución será su versión compilada. Para la consulta, se utilizará la ontología de los conceptos del lenguaje Java, mientras que la solución utilizará individuos que serán instancias de los conceptos de la ontología relativa a la JVM. En realidad, en nuestra representación ontológica de los casos, la descripción y su solución están *fusionadas* en la base de casos. Esto era lo que nos permitía, en la sección 6.6 mantener *dependencias* explícitas entre la consulta y la solución para poder realizar la adaptación.

En cualquier caso, esta aproximación nos proporciona múltiples ventajas. Quizá la más significativa es que podemos tener varios casos para la misma consulta, es decir más de una forma de compilar el mismo código. Por ejemplo, decíamos que el uso en una expresión del número `4` podía compilarse o bien con `bipush 4`, o bien con `iconst_4`, una instrucción en el que el operando está implícito[12]. Para permitir ambas compilaciones, tendremos dos casos. El primero será específico para el número entero `4`, tal y como puede verse en la figura 6.11a. En ella aparece *simultáneamente* la consulta y la solución. La consulta contiene a los individuos de los conceptos `IntegralLiteral` y `Type`, junto con el valor número `4`. Sólo si *los tres* están en la consulta, el sistema CBR que implementa el compilador proporcionará como solución ese caso. La solución, naturalmente, está compuesta por un bloque `JVMBlock` que tiene una única instrucción, individuo del concepto `iconst_4`. Es nece-

[12] En realidad esto ocurre con los enteros entre `-1` y `5`.

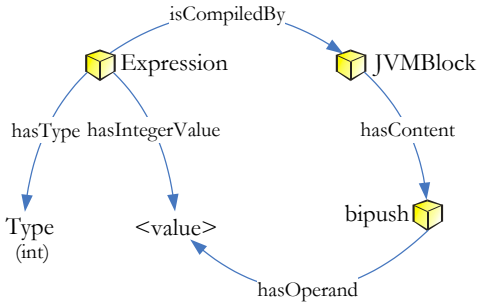


Figura 6.12: Generalización del caso de la figura 6.11b

sario resaltar que estos dos conceptos (`JVMBlock` e `iconst_4`) pertenecen a la ontología relativa a la JVM, no a la del lenguaje de programación Java.

Por otro lado, tendremos el caso general que permite compilar el uso de cualquier constante entera, tal y como se muestra en la figura 6.11b. La descripción del caso deja abierto el valor numérico específico de la constante. Además, hacemos explícita la relación entre la *descripción* del caso y su *solución* a través, precisamente, de dicho valor.

Cuando el sistema se encuentre ante la disyuntiva de saber si el estudiante ha compilado bien o no la estructura OWL del ejercicio que representa a la constante entera `4`, lanzará la consulta al sistema CBR que implementa el compilador basado en casos. Para eso, envía como consulta la estructura formada por el individuo `IntegralLiteral`, `Type` (`int`) y el valor numérico `4`. Desde el punto de vista del sistema, *hay dos soluciones*, pues el caso general *también se aplica*. De esa forma, cualquiera que sea la alternativa que haya escogido el estudiante de ambas, el sistema la reconocerá como válida.

La segunda ventaja de este modo de mantener la información de traducción es que podemos aprovechar la riqueza semántica de la ontología, en concreto sus relaciones "es un", para *generalizar* los casos. Por ejemplo, la ontología sobre Java dispone del concepto `Expression` para representar una expresión. De ella heredan dos conceptos, que permiten diferenciar una expresión final[13] (`Terminal`) de una compuesta (`OperatorExpression`). A su vez, cada uno de ellos tienen sus propios subconceptos. Por ejemplo, `Terminal` es antecesor de `IntegerLiteral` (que ha aparecido en varios ejemplos), y `OperatorExpression` lo es de `AddExpression`.

Dicho esto, cualquiera de los dos casos de la figura 6.11, puede ser generalizado si en lugar de exigir un individuo del concepto `IntegerLiteral` indicamos que se aplica sobre cualquier individuo de `Expression`. Es decir, podemos retirar de nuestro sistema CBR de compilación el caso representado

[13] Llamamos expresión final a aquella que no tiene operadores. Dentro de este grupo está el acceso a variables o el uso de valores constantes.

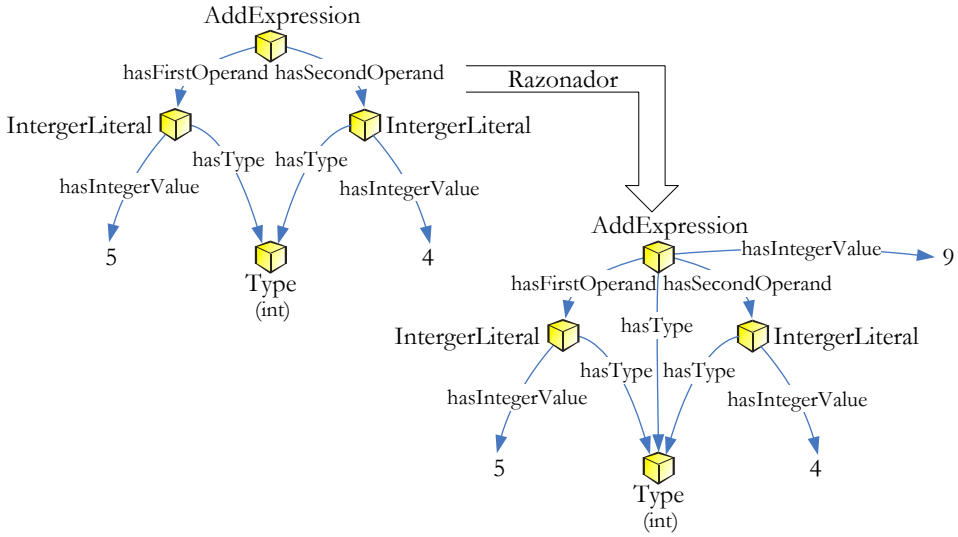


Figura 6.13: Razonamiento para optimizar la compilación de una expresión

en la figura 6.11b y dejar el de la figura 6.12 dado que este último *subsume* al primero. Esto mismo es posible con el caso en el que se utilizaba `iconst_4`.

Con este cambio, en principio parece que no hay una diferencia significativa. Sin embargo, dado que el nuevo caso es más general, puede aplicarse *en más situaciones*. Tan sólo será necesario disponer de un *razonador* que sea capaz de extraer información oculta de la estructura del programa en Java. Por ejemplo, ante el código de ejemplo que utilizamos previamente:

```
5 + 4
```

la herramienta `java2owl` habrá construido una estructura similar a la mostrada en la mitad izquierda de la figura 6.13. Si el razonador sobre esa estructura es lo suficiente perspicaz como para completar la información del individuo de tipo `AddExpression`, se añadiría información no sólo del tipo, sino también del *valor*, dado que éste puede obtenerse directamente a partir de la información de los operandos. El resultado es una estructura en la que se indica que el individuo del concepto `AddExpression` *tiene como valor* el número entero `9` (resultado de la suma).

Lo interesante de esto es que, dado que `AddExpression` *es un* subconcepto de `Expression` y no sólo tiene tipo entero sino que además tiene asociado un valor, el caso genérico que añadimos a nuestro compilador CBR en la figura 6.12 *también encaja*. Por tanto, el sistema *dará por buena* la compilación de dicha expresión con un mero `bipush 9`, que constituye la tercera de las alternativas que nos mostraba la tabla 6.1.

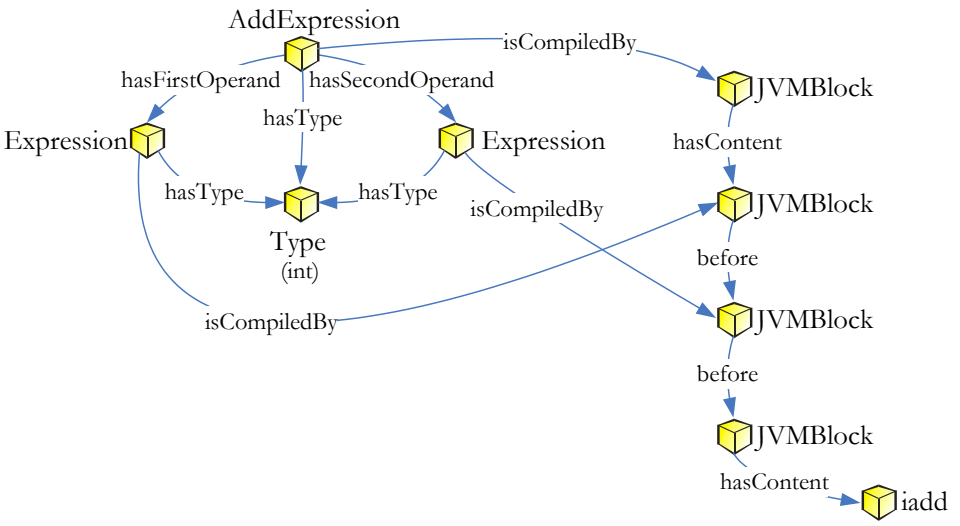


Figura 6.14: Caso para la traducción de una expresión de suma de enteros

Obviamente, el compilador CBR también tendrá los casos que permiten la traducción de las operaciones de suma que *no* se pueden resolver en tiempo de compilación. La figura 6.14 muestra uno de ellos, en el que el primer operando (individuo del concepto `Expression`) es compilado primero. En este caso, la solución *es parcial*, dado que la compilación de cada uno de los operandos *se queda incompleta*. Se reconoce la existencia de un bloque de instrucciones JVM, pero que deberá ser completado más adelante cuando se analice con más detalle la fisiología del operando concreto.

Añadiendo un caso similar en el que primero se compila el *segundo* operando y luego el primero (aprovechando la propiedad conmutativa de la suma), seríamos capaces de reconocer como válidos *todos* los modos de compilación planteados en la tabla 6.1 e incluso las variaciones debido al uso de `iconst_?` en lugar de `bipush`.

Las ideas expuestas sobre el sistema CBR para compilación son también aplicables a construcciones más complejas, como `if` o bucles. En ese caso, la solución podría necesitar hacer explícita la necesidad de etiquetas de código, que se marcarían como individuos de un "tipo" auxiliar de instrucción de la JVM. Así, la compilación del `if`, por ejemplo, supondría la compilación de su expresión condicional *negada*, una instrucción de salto al caso `else`, el bloque resultado de la compilación del caso *sí*, un salto incondicional al final de la estructura, la etiqueta del caso `else`, la compilación del caso *no* y la etiqueta de final. Ahora los *bloques* de instrucciones JVM que *no* son compilados con el caso `if` constituyen *los huecos* que el estudiante iba abriendo a lo largo de la compilación (y ejecución) del programa para aquellas porciones de código

que no eran alcanzadas. Esto termina, finalmente, de cerrar el problema del paralelismo entre compilación y ejecución, dado que el sistema es capaz de seguir la pista a los *huecos* y saber a qué porciones de código fuente (no compilado aún) pertenecen.

Para poder poner en práctica todo esto, es necesario que el sistema sea capaz de pasar de la representación en OWL del *código compilado* a su versión en texto, y viceversa. Es un requisito similar al que ya teníamos sobre el *código fuente* en Java, pero ahora enfocado al código objeto, dado que también lo representamos de manera estructurada a través de individuos OWL.

Al igual que hacíamos con el sistema CBR que nos recuperaba ejercicios, podemos aquí aplicar una pequeña comprobación *general* sobre el funcionamiento del sistema. Al fin y al cabo, lo que tenemos es un *compilador* o, mejor dicho, un comprobador de compilaciones correctas. Para probar que, al menos parcialmente, funciona podemos coger cualquier programa escrito en Java, compilarlo con `javac` y pedirle a nuestro sistema que nos valide la "solución" que éste suministra. Si notifica que la compilación no es válida, será muestra de que *no* hemos cubierto todos los casos de compilación posible, y será necesario buscar las carencias analizando el modo en el que `javac` ha compilado el código concreto.

Naturalmente, si el sistema CBR da por válida la solución *no* podemos estar completamente seguros de su exhaustividad. Siempre puede darse el caso de que se haya pasado por alto algún modo de compilar el programa, que, por no ser el utilizado por `javac`, no salte a la luz. Esto, de hecho, abre la puerta del *mantenimiento* del propio "sistema experto". La aplicación es capaz de crear ficheros de registro con el modo en el que los estudiantes se desenvuelven y solucionan los ejercicios. Las compilaciones que el sistema determina como *inválidas* pueden ser analizadas a posteriori (o, de manera más realista, por sugerencia de los propios usuarios) para determinar si era el estudiante quien efectivamente se equivocó o ha sido el programa el que ha realizado un análisis inválido. Si la causa es esta última, resulta muy sencillo de solventar pues basta con añadir un caso adicional a nuestro compilador CBR. De hecho, esta es una de las grandes ventajas de este tipo de sistemas cuando se comparan con sistemas expertos tradicionales basados, por ejemplo, en reglas. Las interrelaciones de las reglas puede hacer que resulte muy complicado añadir nuevas situaciones específicas no planteadas de antemano. Sin embargo nosotros tenemos la opción de añadir tantos casos como se necesiten, especificándolos con el grado de detalle que sea necesario (navegando por la jerarquía de conceptos de la ontología y con tantas relaciones como sean necesarias) sin que aparezcan efectos laterales no anticipados.

6.9. Comunicación con el estudiante

En JV^{2}M 1 la comunicación con el estudiante se realizaba a través de un diálogo construido gracias a las explicaciones contenidas en las tres partes del conocimiento y a las preguntas de enlace existentes entre ellas. De hecho, esta fase de comunicación era la gran beneficiada del costoso trabajo de autoría (especialmente en los ejercicios) que los expertos del dominio tenían que realizar. Al estar todo el texto "enlatado" en el conocimiento del sistema, hacer uso de él resultaba muy poco costoso.

En JV^{2}M 2, sin embargo, estamos utilizando un modelo de conocimiento más sofisticado, que se "aleja" del usuario, y profundiza en las necesidades que *el sistema* tiene para poder *razonar* con él. Esto tiene la desventaja de que es necesario una *generación* de explicaciones que, hasta ahora, hemos ido dejando de lado.

De manera general, la primera gran diferencia entre las dos versiones de JV^{2}M es que ahora el sistema *deja de ser proactivo*. Como ya vimos, esto es una decisión de *diseño*, pues queremos que el usuario explore, se equivoque, se dé cuenta de sus errores y *solicite* la vuelta a una situación anterior válida. Si el sistema proporciona al usuario ayuda no solicitada, este requisito no se cumpliría.

No obstante, esto *no impide* que podamos proporcionar información indirecta, aprovechándonos de la riqueza del lenguaje de los videojuegos. Naturalmente, el sistema *sabe* si el estudiante está o no yendo por un mal camino. De hecho, al detectar la primera desviación debe guardar el último estado correcto para poder volver a él cuando el estudiante lo pida. Por tanto, es posible añadir en JAVY *comunicación no verbal* (movimientos o gestos) que indiquen su desacuerdo. Naturalmente, estas muestras de preocupación o incluso enfado *no* deben proporcionarse siempre, para evitar que el jugador abuse de la información y la aproveche en su beneficio. En lugar de eso, JAVY podría realizar movimientos de disgusto pasado un tiempo desde el primer error, o incluso realizar gestos de duda cuando el estudiante esté realizando una compilación *correcta* pero subóptima. Otras reacciones posibles son muestras de impaciencia cuando el jugador contrario va mejor que nosotros, o de apremio cuando se nos está acercando.

También es posible indicar al usuario una compilación reiteradamente incorrecta mediante el aspecto general del entorno. Si el estudiante parece ignorar las pistas que JAVY le proporciona con sus movimientos, el sistema puede ser un poco más explícito girando hacia una visión más oscura del entorno, con ácido chorreando de las paredes, luces parpadeantes, o puertas que chirrían. El objetivo último es *no interrumpir* al jugador para que no se pierda la sensación de inmersión, pero aprovechar la riqueza de los videojuegos para avisarle de manera indirecta que lleva tiempo encadenando errores.

En cualquier caso, cuando el usuario sea consciente de que algo va mal es de esperar que se acerque, por propia iniciativa, a JAVY a pedir explicaciones. El modo de comunicarse con él sigue siendo basado en preguntas y respuestas, pero las preguntas ya *no* están prefijadas. En lugar de esto, el sistema *interpreta* el texto *escrito* por el usuario para averiguar qué quiere saber. Ante preguntas del estilo "¿Qué ocurre?" o "¿Dónde me he equivocado?" la aplicación debería ser capaz de *explicar el error*. Con la información proporcionada por el análisis de la solución únicamente sabemos que el usuario *se ha equivocado*, pero la razón que podemos esgrimir es que no tenemos en el compilador CBR ningún caso que demuestre que la alternativa llevada a cabo por el estudiante es correcta. La solución, por tanto, sería extraer la forma correcta de compilar la porción de código en la que se ha equivocado y explicársela.

Sin embargo, la aproximación basada en casos del compilador nos permite la posibilidad de añadir también *compilaciones incorrectas*. Por ejemplo, para la compilación de las expresiones de suma enteras (figura 6.14), podríamos crear una compilación incorrecta en la que en lugar de utilizarse `iadd` se utilice `fadd`. Naturalmente, hay que anotar dicho caso como *inválido*. Pero, además, podemos asociarle una explicación textual del estilo "En esta expresión, los dos operandos son enteros, por lo que la instrucción de suma que se use debe ser para enteros, no para reales". Si el sistema determina que el alumno ha cometido un error, le dejará seguir sin interrumpirle, pero recordando el punto en el que se desvió y, si la base de casos es lo suficientemente rica, siendo capaz de *explicar por qué*. Ante la pregunta inquisitiva del estudiante de qué está haciendo mal, bastará con recuperar dicha explicación.

Esta idea de asociar explicaciones a casos de compilación inválidos puede hacerse también para los casos *correctos*, en cuyo caso la explicación irá encaminada a describir por qué lo es. Esto puede utilizarse a modo de información a posteriori si el sistema determina que el estudiante ha utilizado una solución *no óptima* (por ejemplo, en el número de instrucciones finales). Si el sistema encuentra un modo mejor de compilación del mismo código, puede aprovechar la explicación de los casos usados en esa solución alternativa más eficiente para animar al estudiante a utilizarlas en el futuro.

Las explicaciones añadidas a los *casos de compilación* proporcionan información en lo referente al proceso de *traducción* de código fuente en código objeto. En JV^{2}M 1 también eramos capaces de proporcionar información sobre los pasos que debían ejecutarse para realizar la instrucción actual, algo que ha dejado de tener sentido con la desaparición de las microinstrucciones. Sin embargo, sigue resultando interesante que el sistema sea capaz de proporcionar información sobre lo que hace cada *instrucción*, o sobre las diferentes partes del entorno virtual, tal y como también era capaz de hacer JV^{2}M 1 .

Para cumplir este objetivo, hemos utilizado CBR textual. En realidad,

por CBR textual se entienden varias cosas dependiendo del contexto. Por ejemplo, entran dentro del paraguas de sistemas de CBR textual los *filtros de spam* que a partir de una *base de casos* de ejemplo con mensajes (texto) previamente *clasificados* son capaces de determinar si un mensaje nuevo es o no correo no deseado.

También se considera CBR textual aquel que recibe una serie de textos que *analiza* y *estructura* de una manera adecuada en función del dominio, para realizar recuperación sobre esa estructura. Ejemplos típicos son sistemas de recomendación de restaurantes, o de búsqueda de pisos, en los que el aspecto de los textos (anuncios) es muy similar y pueden analizarse para crear casos estructurados con su información.

Nosotros usamos la vertiente del CBR textual que utiliza técnicas de IR (*Information Retrieval*, recuperación de información) para *realizar búsquedas*. En concreto, se mantiene un grupo de textos que se supone responden a diferentes consultas, y se recupera uno u otro en función de la pregunta realizada por el usuario. Para eso, se compara la consulta con cada uno de los casos textuales mantenidos por el sistema, y se devuelve el "más parecido". La definición de "el más parecido" es variable dependiendo de la *función de similitud* que se utilice. Una de las más habituales (y la que nosotros usamos) es la *similitud del coseno*. A grandes rasgos, consiste en asumir que los textos son "puntos" colocados en un espacio multidimensional con tantas dimensiones como palabras diferentes aparezcan en todos los textos, de modo que la similitud entre dos textos será proporcional a la distancia entre los dos puntos que los representan en ese espacio. Para mejorar el acierto del sistema suelen retirarse en las comparaciones las palabras sin semántica (como artículos y preposiciones), se utilizan las *raíces* del resto de palabras (para, por ejemplo, considerar igual "compilar que "compilé"), se intentan reconocer sinónimos o corregir faltas de ortografía.

Utilizamos CBR textual tanto para responder preguntas sobre la máquina virtual como sobre el entorno. Para el primer tipo de preguntas, hemos incluído en el sistema textos extraídos de la documentación oficial de la JVM, de modo que disponemos de las descripciones oficiales de cada una de sus instrucciones (figura 6.15). Por otro lado, se han desarrollado textos explicativos sobre el entorno en función de la metáfora y el diseño del videojuego. Para implementar el CBR textual, se ha hecho también uso de jCOLIBRI, apoyándonos esta vez en una extensión para, precisamente, CBR textual descrita en Recio et al. (2005).

Conclusiones

En este capítulo hemos terminado de demostrar la versatilidad del modelo de fusión de sistemas educativos y videojuegos que se describió en el capítulo 4. Se ha descrito una segunda instanciación de dicho modelo, variando la

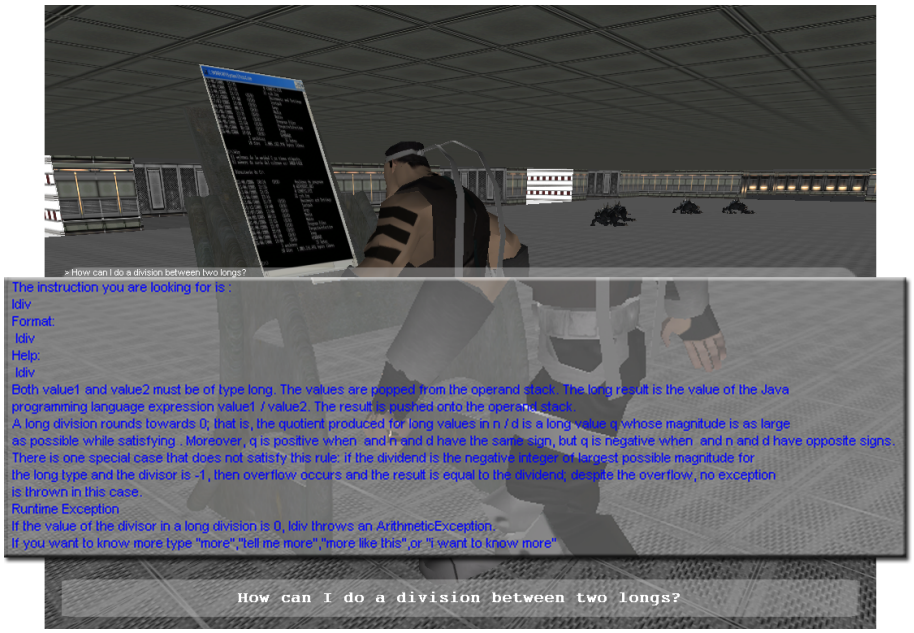


Figura 6.15: JV^{2}M 2: ayuda sobre la JVM

primera (detallada en el capítulo anterior) en sus dos ejes principales: diseño (parte lúdica) e implementación de los módulos de tutoría (parte educativa).

Ambos aspectos han sufrido un giro muy significativo y, a pesar de todo, la aplicación sigue encajando perfectamente en la estructura del ciclo de ejecución esgrimido. Esto pone de manifiesto la generalidad y validez del modelo, que no se encuentra atado a ningún género videojuego o modo de implementación de los aspectos pedagógicos del sistema.

También se ha descrito el uso de sistemas de razonamiento basado en casos ricos en conocimiento para cubrir las labores de tutoría del sistema. Creemos firmemente que el uso de KI-CBR es ideal en aquellos dominios complejos en los que resulta complicado *formalizar* las labores de creación al vuelo de ejercicios o resolución de los mismos.

En efecto, en dominios mínimamente complejos puede resultar demasiado caro (si no imposible) utilizar aproximaciones más formales como sistemas expertos basados en reglas. La alternativa habitual ante esta situación es, naturalmente, hacer uso de casos. Sin embargo, el uso de CBR (sin soporte alguno de conocimiento terminológico del dominio) puede en realidad suponer un alto coste de autoría tal y como quedó patente en el capítulo anterior. Como el sistema carece de conocimiento "general" que pueda ser reutilizado entre diferentes ejercicios, los autores del material se ven en la obligación de añadir una y otra vez el mismo tipo de explicaciones a los casos con los que pueblan el sistema.

El uso de KI-CBR alivia el problema de la autoría extrayendo de los casos aquella información sobre el dominio que pueda aplicarse a lo largo y ancho del sistema. Esta información adicional da soporte a las etapas del ciclo CBR, que pueden ahora mejorarse (especialmente la adaptación) gracias a ella.

En nuestro caso optamos por el uso de *ontologías*, que aportan semántica, relaciones y restricciones a los diferentes componentes de los casos. El enriquecimiento tanto de las descripciones como de las soluciones conseguidas gracias a ellas nos permite *razonar* con comodidad tanto *internamente* al sistema CBR como de manera externa una vez que tenemos su respuesta.

En el caso de las aplicaciones educativas que siguen nuestro modelo, es necesario que el sistema sea capaz de realizar varias labores que, aunque relacionadas, pueden mantenerse relativamente separadas. De hecho, cada una de ellas puede resolverse gracias a un sistema CBR diferente tal y como hemos mostrado aquí. De nuevo, la ventaja de utilizar la alternativa rica en conocimiento nos permite que, aunque cada subsistema CBR tenga *su propia base de casos*, el conocimiento general del dominio (las ontologías) *sean comunes* a todos ellos, haciendo las veces de "lenguaje común" gracias al cual las diferentes etapas pueden entenderse y coordinarse.

Capítulo 7

Conclusiones y trabajo futuro

Olvidábaseme de decirte que esperes el Persiles, que ya estoy acabando, y la segunda parte de Galatea.

Don Quijote de la Mancha II,
Miguel de Cervantes

Resumen: En este último capítulo se hace un recorrido a modo de resumen por todos los anteriores, recalcando los aspectos más importantes descritos y las que consideramos contribuciones más destacadas.

7.1. Conclusiones

Ya fuera debido a los tímidos intentos de principios del siglo XX de desarrollar aparatos mecánicos para *automatizar la enseñanza*, o a las presiones por la necesidad de formación en tiempos de guerra, desde épocas muy tempranas de la Historia de la Informática se fraguó la idea de utilizar los ordenadores para enseñar.

Aunque hay diferentes aproximaciones para conseguir este objetivo, existen dos grandes *familias* de sistemas, que aquí hemos llamado sistemas *basados en marcos* y sistemas *de tutoría inteligentes*.

Los primeros, que llegaron antes, siguen una aproximación *conductista* del aprendizaje. Bajo este enfoque pedagógico, el conocimiento que se desea enseñar se *trocea* en porciones (marcos) que se relacionan entre sí por medio de enlaces. Guiado por los resultados de *tests* sistemáticos realizados a los estudiantes, el sistema decide el recorrido a lo largo de todos esos marcos, proporcionando un camino adaptado al estudiante para potenciar el aprendizaje.

La implementación de estos sistemas se basa en una separación explícita del *contenido* del curso (los marcos u *objetos de aprendizaje*) realizado por expertos del dominio, y el *motor* de la aplicación, que muestra ese contenido al estudiante y le plantea las preguntas propuestas por sus creadores. Por desgracia, aunque dicho "motor" sea capaz de soportar complejas relaciones entre los objetos de aprendizaje, en la práctica los cursos creados casi siempre terminan siendo lineales. Esto es debido al fuerte coste de la autoría del contenido, que supone una tarea muy laboriosa y que requiere mucho cuidado para mantener la coherencia, especialmente cuando el número de ramificaciones se incrementa.

Por otro lado, la cantidad de posibilidades en lo referente a interacciones con el usuario está bastante limitada. Normalmente no van más allá de meros tests, en los que el estudiante tiene que escoger entre una de varias respuestas previstas de antemano. Pedagógicamente esto impone sus propias carencias, restringiendo el tipo de dominios para el que son aplicables. La parte positiva es que la interacción hombre-máquina es muy sencilla y no requiere apenas tiempo de aprendizaje.

El segundo gran grupo de sistemas educativos lo forman los tutores inteligentes, en los que, en lugar de *cablear decisiones* en los enlaces entre marcos, se incrusta *conocimiento explícito* con el que el sistema puede *razonar*. Estos sistemas importan ideas del área de la Inteligencia Artificial, alivian la labor de autoría de *contenido* y, a pesar de todo, son capaces de proporcionar buenas respuestas a los estudiantes gracias al razonamiento. Además, se desplazan a una aproximación *cognitivista* de la enseñanza, y proporcionan modos de interacción más ricos. Esto podría hacer más costoso el aprendizaje del *uso* del sistema al salirse de los familiares interfaces WIMP pero, superado éste, la enseñanza del dominio puede mejorar sustancialmente.

Naturalmente, todas estas ventajas tienen un precio. Es necesario incrustar en el sistema *conocimiento declarativo*, por lo que el problema de la autoría migra de la creación de *contenido* a la *representación de ese conocimiento*. Ésta ha resultado ser una de las mayores dificultades de la Inteligencia Artificial, pues los expertos humanos encuentran muy difícil hacer explícito su conocimiento utilizando un lenguaje que los ordenadores puedan entender.

En otro orden de cosas, la entrada en la "era digital" está revolucionando el modo en el que vemos el mundo. Hoy ya no tiene tanto valor *práctico* ser capaz de dividir rápidamente, o tener una gran enciclopedia en casa. El entorno productivo también ha cambiado, y las empresas reclaman de sus trabajadores aptitudes (y no sólo conocimientos) que no se enseñan en la escuela.

Por otro lado, los videojuegos han cambiado la forma de ocio de los jóvenes. El esfuerzo para terminar un videojuego es cada vez mayor, mientras que los responsables de los planes de estudio simplifican una y otra vez

los currículos de las asignaturas para conseguir esconder el fracaso escolar. Ante esta gran incoherencia asalta la duda de si no sería posible incrustar la motivación intrínseca de los videojuegos en los sistemas educativos.

Bajo esta idea surgen los videojuegos educativos. En realidad la posibilidad de utilizar videojuegos para enseñar surgió al mismo tiempo que los propios videojuegos, y a lo largo del tiempo se han hecho muchas cosas, principalmente enfocados a niños al ser un público más fácilmente manipulable. No obstante, si intentamos encasillar este tipo de videojuegos en los sistemas educativos anteriores, encajan con la filosofía de funcionamiento de los sistemas basados en marcos. Hay muy pocos ejemplos de videojuegos que realmente *enseñen* y *guíen el aprendizaje*. No suelen proporcionar ayuda, y se asientan sobre una laboriosa autoría de contenido.

Tomando como sustrato toda esta situación, este trabajo propone dos ideas principales:

- Añadir a los videojuegos educativos aspectos de tutoría más elaborados, tal y como se encuentran en los tutores inteligentes. Para eso, plantea un marco de trabajo en el que se fusiona un ejercicio y un nivel de juego. Alrededor de él se colocan los diferentes tipos de conocimiento que tradicionalmente se han incluido en los tutores inteligentes y se describen las diferentes decisiones que deben tomarse para su desarrollo.

- Hacer uso de razonamiento basado en casos *rico en conocimiento* (KI-CBR) como método de representación de la información que el sistema debe guardar para llevar a cabo sus labores de tutoría. Esto se sienta a medio camino entre las necesidades de *autoría* existentes en los sistemas basados en marcos (que requieren *contenido*) y de los tutores inteligentes (que requieren *conocimiento* con el que razonar). Al utilizar razonamiento basado en casos resulta mucho más sencillo "extraerlo" de los expertos por ser conocimiento *episódico*. Al mismo tiempo, se hace explícito cierto conocimiento general del dominio que puede reaprovecharse entre los casos ahorrando repetición.

El modelo de creación de videojuegos educativos, que fusiona un nivel de juego y un ejercicio, es lo suficiente general como para que puedan encajar en él gran cantidad de sistemas. En concreto, deja abierto el dominio, la representación de conocimiento (e implementación de los "módulos" que lo usan) y el género del videojuego.

Para ponerlo a prueba, se han descrito dos instanciaciones diferentes, en las que se ha hecho variar tanto el diseño del videojuego como la representación del conocimiento. Por comodidad, el dominio no se ha modificado (para no requerir expertos), pero podría, naturalmente, también variarse.

Por su parte para poner a prueba la hipótesis de que el uso de KI-CBR es bueno, la primera instanciación usa razonamiento basado en casos tradicional, y la segunda migra hacia una representación donde, aunque sigue habiendo casos, hay una aparición explícita de conocimiento general del dominio por medio de ontologías. Con esto mostramos que esta segunda instanciación que usa KI-CBR dispone de unas capacidades de tutoría más avanzadas sin incrementar la labor de autoría.

A modo, pues de resumen, las aportaciones más importantes de este trabajo son:

- Se ha analizado el ámbito de las aplicaciones educativas, enmarcándolas en las diferentes tendencias pedagógicas que las dan soporte a la vez que se prestaba atención al modo de implementarlas.
- Se ha hecho un recorrido por los videojuegos educativos, destacando sus ventajas en lo referente a la motivación, y descubriendo la carencia de componentes de tutoría de los videojuegos educativos disponibles en el mercado.
- Se ha propuesto un modelo de videojuegos educativos, consistente en fusionar un nivel de juego y un ejercicio del dominio. Se han estudiado los diferentes ejes de decisión que permiten añadirles las características educativas encontradas en los tutores inteligentes.
- Se ha propuesto el uso de KI-CBR para la creación de los diferentes tipos de conocimiento necesarios para dichos sistemas, destacando las ventajas que tiene quedarse con lo mejor de la representación explícita de conocimiento y de la mera generación de contenido.
- Se han propuesto *dos* instanciaciones del modelo, que, dejando fijo el dominio enseñado, modifican el diseño del videojuego y el modo de representación del conocimiento. Esto pone a prueba la generalidad del modelo, así como permite comparar la aproximación de representación KI-CBR con la CBR pura, que queda en clara desventaja.

7.2. Trabajo futuro

El trabajo descrito en este documento se ha desarrollado a lo largo de varios años, en los que se han explorado diferentes caminos y alternativas para el desarrollo de videojuegos educativos. Hemos profundizado en cuestiones referentes no sólo a la implementación de los aspectos pedagógicos, sino también sobre la *arquitectura* del software de entretenimiento. Muchas de ellas, aun habiendo sido revisadas bajo el marco en el que se engloba el presente trabajo, han quedado fuera de los objetivos de esta memoria. Otras

cuestiones, sin embargo, nunca han recibido la atención necesaria como para ser dignas de mención, quedando, por el momento, como trabajo futuro.

Por ejemplo, aquí nos hemos preocupado sobre el modo de controlar la dificultad de *los ejercicios*, a partir de la elección de los siguientes conceptos a practicar y posteriormente de un ejercicio que encaje con ellos. Sin embargo, para que un videojuego educativo esté completo también es necesario prestar atención a la dificultad de *la componente lúdica*. Aunque hemos hecho algún tímido intento por adentrarnos en este campo, aún quedan cosas por hacer. Además, constituye un problema por sí mismo, que *no* es exclusivo de los videojuegos *educativos*, sino que resulta de interés para prácticamente cualquier género. En el plano un poco más práctico de esto, hemos realizado algunas pruebas de implementación en JV^{2}M 2 para modificar la dificultad de los personajes del juego controlados por el ordenador, pero son aún demasiado modestas. Creemos que la dificultad de la componente lúdica tiene relación con la sensación subjetiva de dificultad general de todo el sistema. Además, podría entorpecer el proceso de aprendizaje que, al fin y al cabo, es lo que se quiere optimizar. En este aspecto todavía hay trabajo por hacer.

Por otro lado, nos gustaría revisar el modo en el que representamos actualmente el modelo del estudiante, y la forma en la que escogemos los siguientes conceptos a practicar. Siempre hemos tomado el camino más sencillo de utilizar plantillas con las que se toman decisiones relativamente automáticas.

En realidad, el sueño dorado de los tutores inteligentes es disponer de un modelo del estudiante *ejecutable*, que permita *predecir* el resultado de enfrentarle a un nuevo ejercicio. El uso de plantillas queda demasiado lejos de este objetivo.

Por otro lado, en la literatura, suele introducirse la idea del razonamiento basado en casos como algo muy cercano al modo de pensar de los humanos. Al fin y al cabo, nosotros aprendemos por la experiencia, y la solución que dimos a un problema anterior la repetimos una y otra vez sin replantearnos las reflexiones lógicas que nos llevaron a ella en primer lugar.

Ambas reflexiones forman una pareja perfecta con la idea, aún inexplorada, de representar el modelo del estudiante *mediante casos*. En realidad, en el capítulo 4 se mencionó un trabajo en el que, de hecho, implementaban el módulo pedagógico usando CBR. Sin embargo, lo que hacían era mantener casos donde la descripción era *el modelo del estudiante*, y el resultado era *el ejercicio propuesto*, junto con una anotación sobre si el resultado pedagógico fue bueno o no.

Lo que querríamos explorar es una representación del *modelo del estudiante* (no una implementación del módulo pedagógico) usando casos, en concreto aquellos ejercicios que ya ha resuelto y que puede utilizar como base para resolver otros nuevos. Para que esto tenga sentido, necesitamos que

el sistema implemente un módulo que sea capaz de resolver *nuevos ejercicios* a partir de las *soluciones de otros*, pudiendo así "simular" el modo de resolución de ejercicios que, seguramente, realizará el estudiante.

En JV^{2}M 2 nos encontramos en una situación ideal para explorar este nuevo aspecto, dado que, de hecho, ese modo de resolución está intrínseco en el *compilador basado en casos*. La idea es que, una vez decidido el próximo ejercicio a plantear, podremos anticiparnos al resultado si intentamos resolverlo con dicho "compilador" al que hemos alimentado, tan sólo, con aquellos casos de compilación que el estudiante *ha usado ya*. Si este resolutor CBR se muestra *incapaz* de proporcionar una solución, podremos asumir que el estudiante tendrá dificultades con el ejercicio, que deberá sortear haciendo uso de *razonamiento tradicional* o, sencillamente, *sentido común*. De hecho, podríamos indagar más y averiguar qué casos de compilación *desconoce* para ser capaz de llevar a buen puerto el ejercicio, y proporcionarle algún tipo de ayuda previa que le ponga sobre aviso.

Para *actualizar* el modelo del estudiante basta con *añadir* aquellos casos de compilación que demuestre conocer, es decir que haya usado en el último ejercicio. En realidad como la base de casos de nuestro compilador CBR guarda casos *generales*, será necesario añadir al modelo del estudiante las *particularizaciones* que haya utilizado, aprovechando la jerarquía de los conceptos de la ontología. Esto añade un punto adicional a explorar, relativo a cuando asumir que el estudiante ha realizado su propia *generalización* sobre los casos que ya conoce, para realizar un "mantenimiento" de la base de casos que representa su conocimiento. Por ejemplo, si el sistema mantiene en el perfil del estudiante los casos (particulares) de la compilación de una expresión aritmética de suma de enteros, y el de suma de flotantes, muy posiblemente podrá asumir que el estudiante habrá "generalizado" el proceso y también será capaz de sumar enteros y números reales largos (`long` y `double`).

El punto verdaderamente interesante de esta idea para disponer de un modelo del estudiante ejecutable consiste en que puede usarse no para anticipar lo que éste hará con el ejercicio que se le va a plantear, sino para *planificar sobre él*. Con esto, en lugar de utilizar una mera lista de reglas que deciden los siguientes conceptos a partir de una plantilla representando el modelo del usuario, pasaríamos a utilizar un *planificador* que pruebe diferentes alternativas y decida seguir el camino que mejor resultado parezca tener en función de lo que el estudiante sabe ya.

Apéndice A

Ejemplo de ejecución de JV^{2}M

Este apéndice contiene un ejemplo de ejecución de la versión de JV^{2}M descrita en el capítulo 5. Primero se muestra el código fuente del ejercicio, el código objeto (de la JVM) resultante de su compilación, y el modo en el que el usuario debe desenvolverse por el entorno para resolverlo.

A.1. Código fuente en Java

El ejercicio está compuesto por tres sencillas clases. Pretende poner en práctica la creación de objetos y la invocación a métodos heredados de la superclase. Este ejercicio se tendría que resolver una vez que el alumno ha demostrado que comprende la estructura de la JVM y la compilación de las *construciones imperativas* (expresiones, condicionales, etcétera).

La `ClasePadre` tiene un único método que devuelve un valor constante (`0`). Éste será llamado en el programa principal a través de un objeto de `ClaseHija` para comprobar la herencia de métodos:

```
public class ClasePadre {

    public ClasePadre() {}

    public int devuelveCero() {
        return 0;
    }

}
```

La `ClaseHija` añade un `metodo` adicional sin código. No tiene una utilidad especial, tan sólo poner de manifiesto que las subclases pueden tener métodos no disponibles en la clase padre. De hecho, el método no es invocado en ningún momento, por lo que el estudiante *no* tendrá que compilarlo.

```
public class ClaseHija extends ClasePadre {

    public ClaseHija() {}

    public void metodo() {}

}
```

Finalmente, el programa principal crea una instancia de la clase hija, e invoca al método de la padre:

```
public class Main {

    public static void main(String args[]) {
        ClaseHija c;
        c = new ClaseHija();
        c.devuelveCero();
    }
}
```

A.2. Código objeto de la JVM

El código compilado de la clase padre es:

```
Method ClasePadre()
   0 aload_0
   1 invokespecial #1 <Method java.lang.Object()>
   4 return

Method int devuelveCero()
   0 iconst_0
   1 ireturn
```

El de la clase hija:

```
Method ClaseHija()
   0 aload_0
   1 invokespecial #1 <Method ClasePadre()>
   4 return

Method void metodo()
   0 return
```

Y el del programa principal:

```
Method Main()
   0 aload_0
   1 invokespecial #1 <Method java.lang.Object()>
   4 return

Method void main(java.lang.String[])
   0 new #2 <Class ClaseHija>
   3 dup
   4 invokespecial #3 <Method ClaseHija()>
   7 astore_1
   8 aload_1
   9 invokevirtual #4 <Method int devuelveCero()>
  12 pop
  13 return
```

A.3. Descripción de la ejecución

En Java, la ejecución de los programas comienza con el método `main` de la clase de inicio. El cargador de la máquina virtual carga dicha clase, y realiza la llamada al método estático `main`, pasando como parámetro las cadenas de los argumentos de la línea de órdenes que haya podido recibir el programa. Cuando el estudiante va a comenzar a resolver un ejercicio en el entorno virtual, todas las clases estarán ya cargadas, y el contexto de ejecución del método `main` habrá sido ya creado. Por lo tanto, existirá un frame, con la pila de operandos, variables locales y máquina de calcular, y en el barrio de clases estarán creados todos los edificios, cada uno de ellos representando a una de las clases cargadas por la máquina virtual. El avatar del alumno estará situado en el interior del edificio del frame, con los objetos representando a los operandos de la primera instrucción del método dentro de su inventario.

En el ejercicio que estamos resolviendo, la primera instrucción a ejecutar es la de creación de un nuevo objeto de la clase `ClaseHija`, y el alumno tiene en el inventario el objeto virtual representando a dicha clase.

El primer paso es solicitar la creación del objeto. Para eso el jugador sale del edificio del frame y se dirige hacia el barrio de objetos creados, que está vigilado por un portero, llamado *Objy*, que es el encargado, precisamente, de construir nuevos objetos. El alumno le entrega el objeto representando el nombre de la clase de la que crear un nuevo objeto, y *Objy* crea el nuevo objeto proporcionando al alumno un objeto virtual nuevo, que indica la localización del joven objeto en el barrio. Por tanto, la operación en el entorno realizado por el usuario es *Dar a Objy el* ***Nombre de la clase*** (Usar Nom-

bre de la clase con *Objy*). El nuevo objeto es creado, y, de forma automática, *Objy* da un nuevo objeto *Dirección del nuevo objeto* al alumno.

Para terminar la instrucción `new`, el alumno debe apilar la referencia del objeto nuevo. Es decir, entra en el edificio del frame, y *Usa la dirección del nuevo objeto con la pila de operandos*. Esto finaliza la instrucción, y el programa saltará a la siguiente, la instrucción `dup`.

La instrucción `dup` hace una copia del elemento situado en la cima de la pila, para lo que se hace uso de la máquina de calcular situada en el *frame*. El primer paso es desapilar la cima mediante la operación *Coger pila de operandos*. Esto hace que en el inventario del usuario aparezca un nuevo objeto, denominado *Valor original de la cima*. El siguiente paso es hacer una copia de ese objeto. El usuario debe colocar el objeto en la máquina de calcular mediante la operación *Usar valor original de la cima con entrada1 de la máquina de calcular*. Posteriormente, hay que configurar la máquina para que el resultado de la operación a ejecutar sea una copia exacta del objeto situado en su primera entrada. Para eso se manipulan sus palancas, y se establece la operación deseada. Después se *Usa la palanca de cálculo* para poner a trabajar a la máquina, que deposita en su salida la copia. Ambos valores deben ser recogidos con *coger valor original de la cima* y *coger copia de la cima*, objetos que pasarán a formar parte del inventario.

El último paso es introducir las dos copias de nuevo en la pila, con *usar copia de la cima con pila de operandos* y *usar original de la cima con pila de operandos*, con lo que terminamos la ejecución.

El programa descodifica la siguiente instrucción, e introduce en el inventario la representación de sus operandos, en este caso un nombre de clase (`ClaseHija`) y un nombre de método (`ClaseHija()`). La nueva instrucción es la llamada al código del constructor del objeto que estamos creando.

El primer paso es desplazarse al barrio de clases y buscar en él el edificio de la clase `ClaseHija`. Una vez dentro de él, el usuario se encuentra una máquina, protegida por una vitrina cerrada. Esta vitrina se abre si se dispone de la llave adecuada, que no es más que el objeto del inventario representando a la clase en la que estamos. Por tanto, el usuario decide *usar nombre de clase con vitrina*, y la vitrina se abre, a cambio de destruir el objeto proporcionado.

Una vez que el alumno tiene acceso a la máquina de control de la clase, se le solicita a ésta el método a ejecutar. Para eso se *usa el nombre del método con entrada de máquina de búsqueda de métodos* y se *usa la palanca de búsqueda de método*. La máquina absorbe el objeto representando al nombre del método buscado, y devuelve un nuevo objeto representando a su código, en este caso al código del constructor. El usuario *coge el código del método* y se marcha del barrio de clases.

Cercano al edificio del *frame*, hay también un personaje, llamado *Framauro*, al que el usuario le solicita la creación de nuevos *frames*, y la destrucción de los antiguos. Cada *frame* representa un entorno de ejecución, es decir, un

método cuya ejecución aún no ha terminado. Como queremos realizar una llamada al constructor (cuyo código ya tenemos) y aún no se ha terminado de ejecutar el método anterior, tendremos que crear un nuevo contexto de ejecución, es decir, un nuevo *frame*. Para eso el usuario le solicita a *Framauro* su creación, dándole el objeto con el código del método, es decir, *da el código del método a Framauro* (usar el código del método con *Framauro*).

Framauro creará un nuevo *frame* que se convierte en el actual. Para eso, en primer lugar analiza la información contenida en el código del método y ve que el método a llamar espera un parámetro, que extrae de la pila de operandos del *frame* antiguo (encontrando la dirección del objeto recién creado). Después oculta el *frame* antiguo, haciendo que desaparezca hundiéndose en el suelo, y crea uno nuevo, que cae del cielo ocupando la posición del anterior. Para acabar, introduce en la lista de variables locales del nuevo *frame* los parámetros que ha extraído de la pila de operandos vieja, en este caso uno solo, representando al objeto creado que ocupará la posición `0` de las variables locales.

Esto finaliza la ejecución de la instrucción, y el programa prepara la ejecución de la siguiente, que será la primera instrucción del constructor de `ClaseHija`.

Esta instrucción es la carga del valor de una variable local, la situada en la posición 0, para cuya ejecución el sistema habrá incluido en el inventario del usuario el objeto *posición 0 de variable local*. El primer paso es hacer uso del dispositivo de acceso a las variables locales del frame. El alumno entra en el edificio que lo representa, y coloca el objeto anterior en dicho dispositivo, es decir *usa posición 0 de variable local con dirección a acceder*. Después, *usa la palanca de lectura* para que el dispositivo haga una lectura, y *coge la variable local leída*, que debe ser introducida en la pila de operandos con *usar variable local leída con pila de operandos*.

La siguiente instrucción es de nuevo una llamada al constructor, que ya hemos realizado y no volveremos a describir.

Por último, la ejecución del constructor termina con la instrucción de vuelta. El sistema se habrá encargado de introducir en el inventario del usuario un objeto representando el valor a devolver. Como no se devuelve nada, el objeto representa en realidad un valor vacío. Al igual que ocurría con la llamada a métodos, la finalización es realizada por *Framauro*. Para eso, el usuario *da el valor a devolver a Framauro*, y éste elimina el *frame* actual (que se eleva y desaparece), y extrae del subsuelo el *frame* antiguo, que reaparece y se convierte en el actual. Además, si el valor a devolver del método anterior no es vacío (que no es el caso), lo introduce en la pila de operandos.

Al finalizar el constructor, se continúa la ejecución en el método `main`, en la instrucción siguiente a su invocación, en este caso una instrucción de almacenamiento de variable local. El sistema habrá introducido en el

inventario el objeto posición 1 de variable local para que el alumno *use la posición 1 de variable local con dirección a acceder* del dispositivo de acceso a las variables locales. Además, hay que proporcionar el valor a escribir. Para eso, se obtiene la cima de la pila con *coger pila de operandos*, obteniendo el objeto *valor a almacenar*, de modo que, posteriormente, se *usa el valor a almacenar con entrada de datos* del dispositivo. Para que la operación de escritura se realice, el último paso será *usar la palanca de escritura*.

La instrucción siguiente es de recuperación de una variable local, que ya se ha explicado. La siguiente es una nueva instrucción de invocación a métodos. El sistema coloca en el inventario del usuario una representación de la localización del objeto usado para la llamada, y una representación nombre del método al que se invoca.

Como en la instrucción de llamada ya vista, el primer paso es conseguir el código del método. Para eso, necesitamos conocer a qué clase pertenece el objeto usado para la llamada. Esa información nos la proporciona *Objy*. El usuario *da el objeto this de la llamada a Objy*, y éste le proporciona una representación de la clase.

Al igual que en la instrucción de invocación al constructor, el siguiente paso es buscar el edificio de la clase, y abrir la vitrina con el nombre de la clase. También ahora le proporcionamos el nombre del método con la esperanza de que nos dé su código. Sin embargo, como se ve en el código fuente, `ClaseHija` no dispone del método `devuelveCero`, por lo que el dispositivo de acceso a métodos nos devuelve una representación del nombre de la clase padre de la anterior y el mismo nombre del método que se le había proporcionado. El avatar del alumno los coge de forma automática, y la vitrina se cierra de nuevo, siendo imposible volver a acceder a ella.

De nuevo estamos en la situación anterior. Tenemos el nombre de una clase y el de un método. Por tanto, repetimos el proceso de buscar el método, yendo al edificio de la clase, y solicitando su código a la "dispensadora de métodos". Lo que estamos haciendo es buscar el código de un método en la clase del objeto, o en sus superclases si el método no se encuentra.

Finalmente, conseguiremos el código del método de `devuelveCero`, y creamos un nuevo *frame* dándoselo a *Framauro*, tal y como hicimos en la llamada al constructor.

La siguiente instrucción a ejecutar será la primera del método `devuelveCero`, que es la de carga de constante. El usuario dispondrá de un valor a apilar en su inventario, y lo único que tendrá que hacer será *usar valor a apilar con la pila de operandos*.

El método termina con una instrucción de finalización que, ahora sí, devuelve algo. El valor a devolver está almacenado en la pila de operandos, por lo que el usuario *coge la pila de operandos*, recibiendo en su inventario el valor a devolver. Como ocurrió en el constructor, ese objeto se le da a *Framauro*, que finaliza el método actual y vuelve al anterior eliminando un

frame y obteniendo el antiguo.

Volvemos así al método `main`, cuya siguiente instrucción elimina la cima de la pila. Para eso, el usuario *coge la pila de operandos* consiguiendo el *valor a eliminar*. Para destruirlo, se utiliza de la máquina de calcular, en concreto, se *usa el valor a eliminar con entrada1 de la máquina de calcular* se manipulan las palancas de configuración de la máquina para que la operación a realizar sea de eliminación, y se *usa la palanca de cálculo*, que recoge el valor de la entrada 1 y lo destruye.

La última instrucción que se ejecuta es la de finalización del método `main`, que es semejante a la de la finalización de los constructores ya vista.

Apéndice B

Historia de fondo de JV^{2}M

Este apéndice describe la historia literaria de fondo de JV^{2}M . Hace las veces de guión del videojuego, que justifica su comienzo, la razón del agente pedagógico y de la ayuda en las tareas laboriosas que proporcionan los campesinos.

B.1. Introducción

— Gggggggggrrrrrrrrmmmmmmmm.....

Desentumeces un poco el cuerpo estirándote, todavía con recelo a abrir los ojos por miedo a que la luz te deslumbre y aumente todavía más el dolor de cabeza. Vaya noche. Fiesta, fiesta y más fiesta, bañada en litros de alcohol, sudor y humo. Tratas de buscar tu último recuerdo. Primero la cena, regada con un buen vino que parecía evaporarse. Luego las copas, la discoteca... Había unos ojos; aquellos ojos... con un extraño mirar.

Qué dolor de cabeza.

Un revuelo. Esos ojos, ahora más cerca... La música no te dejaba pensar.

Qué dolor de cabeza.

Un empujón. Una quemadura de cigarro. Y los ojos junto a ti. Extraño color. Sin pupila ni variación. Azul intenso, pero azul liso, perturbador.

Qué dolor de cabeza.

Un momento. ¿Y después? Abres inmediatamente tus ojos y miras alrededor. Una extraña neblina entorpece tu visión. Te prometes que la próxima vez no beberás tanto. Pero sabes que esa próxima vez romperás la promesa... igual que hiciste anoche.

— Maldita sea – farfullas – ¿dónde diablos estoy?

Extraño sitio. ¿Un pajar? ¿¿Un granero?? Será posible. Cuando lo cuentes en el casino nadie te va a creer. Acabar una noche loca en un granero. Pero... pero ¿con quién? ¿Dónde están esos ojos traicioneros?

Qué dolor de cabeza.

La neblina de tus ojos empieza a desvanecerse. Miras con más detalle. ¿Seguro que esto es un granero? Tiene un estilo extraño. Parece falso, un simple decorado. Atrezzo de película de marujas de Domingo a la hora de la siesta.

— Esta vez he ido demasiado lejos. – te dices a ti mismo – Debería replantearme mi vida... si no me doliera tanto la cabeza. Este no parece un buen sitio para conseguir una aspirina.

Un minuto. ¿Has dicho "decorado de película"? Sí, eso es... pero esto tiene pinta de película de dibujos animados. Es extraña la luz, son extraños los colores, es extraña la forma de las cosas, como hechas con bloques. Hay un estilo familiar en el tono de las cosas. Un estilo que se podía ver en el fondo de... de... aquellos ojos. ¿Dónde se han ido?

Esto es imposible. Inicias el movimiento de restregarte la cara y los ojos para intentar espabilarte y analizar con calma la situación pero...

— **¿¿¿¿¡¡¡¡Qué le pasa a mi mano!!!!????**

Miras todo tu cuerpo desesperadamente. ¿¿Qué le ha pasado a tu ropa?? ¿¿Qué le ha pasado a tu piel?? Todo tiene pinta de dibujos animados, de aquellos que salen de los ordenadores... ¡¡incluido tú!! Una idea asalta tu mente.

— ¡¡Por Dios!! – gritas al aire viciado de ese ridículo lugar – ¡¡Un espejo!! ¡Necesito un espejo!

No. No puede estar pasándote a ti. Te conformas con tu reflejo en el "agua" de un barril. Reflejo falso, extraño, irreal. ¿¿Irreal?? ¿Estás seguro? Es un reflejo artificial, sí. Pero no cabe duda. También tu cara se ha convertido en...

— ¡¡¡¡Me he convertido en un dibujo!!!!

Qué **dolor** de cabeza.

— No, no. Aún sigo durmiendo y esto no es más que es una pesadilla. – tratas de autoconvencerte mientras sigues asomado al interior del barril.

Para demostrártelo, sólo se te ocurre dar un cabezazo al tonel con la firme certeza de que despertarás junto a ella, la dueña de aquellos ojos. Pero...

Qué...

...**inmenso**...

...**dolor**...

...de...

...cabeza.

Vaya luces. Eso no resuelve nada. Sales del granero a toda prisa, como intentando huir de ti mismo, de un lugar que no comprendes, de un pasado que no recuerdas.

Fuera, todo es dibujo. Campos con horizontes fingidos, donde de vez en cuando se perfila algún grupo de campesinos trabajando bajo un Sol embustero. Algunas casuchas pueblan aquel esbozo de la realidad, construcciones imposibles fuera de los mundos de fantasía.

Tu desesperación te lleva a hundirte en un mar de trigo, en busca de una de esas figuras de movimientos mecánicos que labran la tierra, en busca de una explicación que habrías preferido no oír...

B.2. La horrible historia de YOGYAKARTA

A mitad de tu camino hacia el desconocido, el pensamiento de que, quizá, el campesino sea hostil a tu presencia hace que te arrojes rápidamente al suelo para esconderte entre el trigo.

Aquel impulso te salvó la vida.

Sin duda, ya estaban buscándote.

Te acercas al aldeano, agazapado entre las espigas. No es fácil moverse dentro de un cuerpo dibujado de movimientos espasmódicos, especialmente si hay que hacerlo de forma silenciosa. Te colocas tras su espalda y, con un rápido movimiento, saltas sobre él y le tiras al suelo, inmovilizándole para poder interrogarle sin que los demás te vean. Al mirarle a la cara, sientes una sensación extraña. Tiene una difusa mueca de pánico, disuelta en un rostro inexpresivo. De repente, te asalta una duda.

— ¿Sabes hablar? – susurras, tras vencer tus prejuicios de lo ridículo de hablarle a un estúpido dibujo.

El campesino asiente con la cabeza. Tienes la mano tapándole la boca para que no grite. Casi le estás asfixiando... si es que necesita respirar.

— ¿Vas a gritar?

Niega con la cabeza. Poco a poco le sueltas. Parece haber dicho la verdad.

— Maldita sea, qué susto me has dado. Tú no eres un *Hodi*, ¿no? Eres nuevo aquí, ¿verdad? Este no es sitio seguro para hablar... a estas horas andarán buscándote y vendrán aquí. No tienes demasiado tiempo. Ven conmigo. ¡Rápido! Y no se te ocurra asomar la cabeza.

¿¿Seguir a un dibujo?? No pareces tener muchas más alternativas. Y no ha chillado. Te arrastras detrás de él. Te guía a una trampilla oculta, que deja a la vista un escondrijo bajo tierra, como un túnel. Entráis.

— Bien. Aquí estamos seguros. Los *Hodi* no conocen estos sitios porque los hemos excavado nosotros mismos. Podemos hablar sin preocuparnos. Estarán demasiado atareados buscándote como para darse cuenta de que yo he desaparecido. ¿Cuánto sabes de YOGYAKARTA?

— ¿*Hodis*? ¿YOgigA..ta? ¿De qué estás hablando? No tengo ni pajolera idea de qué está pasando aquí.

— YOGYAKARTA. Así se llama la aldea en la que estamos. Ya veo. Ponerte al día va a requerir bastante tiempo. Lo primero. ¿Cómo te llamas?

— Mortando

— Está bien. Yo me llamo Wadet. Estoy seguro de que tu último recuerdo de la pasada noche son unos ojos azules. A todos nos pasa igual. Una noche llegan, y al día siguiente te despiertas en un granero, en *el* granero. Tú has tenido suerte. Has huido antes de que los hodis te encontraran.

— Otra vez esa palabra... "*hodis*" Ya la has dicho varias veces. ¿Quiénes son?

— Tranquilo, todo a su debido tiempo. A todos nosotros, a los que has visto en el campo, nos cogieron el día de nuestra llegada. Y nos convirtieron en campesinos. El primer día aquí es difícil si no consigues escapar. Todo es dibujo, y ellos te manejan a voluntad, a si es que no les resulta difícil transformarte. Tú has sido afortunado. Una vez que sepas la verdad, no podrán hacerte cambiar, aunque pueden encerrarte, cómo le hicieron a él.

— ¿A él?

— ¡Ey! ¡¡Tú has venido a sustituirle!! ¡¡¡Claro!!! ¿Cómo no me había dado cuenta? Has conseguido escapar, escondido aquí podrás evitar que te transformen, y con un poco de suerte ¡podrás ser nuestro libertador!

Aquel dibujo estaba delirando. ¿Cómo ibas a salvarlos tú y de qué?

— Un momento. ¿De qué estás hablando? ¿Dónde estamos? ¿Quiénes son esos *hodis* de los que tanto hablas? ¿Quién es *él*? ¿¿Por qué soy un dibujo??

— Nadie lo sabe a ciencia cierta, y todo lo que te voy a contar es el resultado de nuestras propias conclusiones. Antes de llegar aquí, todos, salvo los *hodis*, teníamos una vida. Una vida real, no de dibujo, igual que la tenías tú hasta que te has despertado en el horrible granero. Cómo llegamos aquí no lo sabemos. De lo que estamos seguros es de que todos tenemos una cosa en común: odiamos los ordenadores. Y estoy seguro de que tú también.

— Sssssí – balbuceas.

— Todos nosotros habíamos, de forma independiente, decidido dejar de lado la informática en nuestra vida cotidiana porque consideramos los ordenadores unos cacharros fríos, e inútiles, que odiamos porque no entendemos. Yo era arquitecto y aún hacía todo mi trabajo de forma manual. Un ordena-

dor no puede ser la herramienta para hacer arte. Entre los campesinos que has visto, hay más arquitectos como yo, algún escritor que ni siquiera ha cambiado su pluma por una máquina de escribir, fotógrafos que consideran que la fotografía digital es un sacrilegio, o secretarias que aún utilizan la taquigrafía. ¿De qué tipo eres tú?

— Vaya... yo soy contable... pero en contra de lo que todos mis compañeros me piden, sigo llevando los libros de cuentas a mano.

— Creemos que esto es un castigo por nuestro odio. Estamos dentro de uno de esos malditos cacharros, Mortando. O al menos, en un mundo creado por uno de ellos. No tenemos ni la menor idea de quién diablos está detrás de todo esto. Quizá alguna empresa del sector a la que la estamos haciendo perder dinero. En cualquier caso, siempre es igual. Unos ojos azules de mujer (o de hombre para ellas) es el último recuerdo que a todos nos queda de la realidad. Después despertamos aquí. No sabemos qué habrá pasado con nuestro cuerpo en el mundo real, si estaremos muertos, si nos considerarán desaparecidos, si podremos volver algún día. Hasta hoy, nadie lo ha hecho.

Qué dolor de cabeza. Una historia imposible, escalofriante y desalentadora.

— Cuando despiertas en el granero, todo tu mundo se desvanece. Es una experiencia que tú recuerdas mejor que yo. Los malditos *hodis* llegan y te confunden. No hay quien lo soporte. Te hacen cambiar. Y te obligan a trabajar el campo de Sol a Sol. Pero son ellos quienes deciden cuando sale y se pone ese Sol. Los *hodis* no son como nosotros. Llamamos *hodi* a todo aquel que no ha venido de fuera.

— Y los *hodis* son... - vuelves a insistir.

— Los *hodis* no son como nosotros. Son seres que siempre han estado aquí, que no han tenido nunca una vida en el exterior de este sitio dibujado, que no son reales. Son los seres que pueblan este mundo recreado, son los ***ho**mo **di**gitalis*. por eso los llamamos *hodis*. Algunos son policías, otros ornitorrincos ninja...

— Y ¿quién es él?

— Es alguien que tuvo tanta suerte como tú y no le cogieron cuando amaneció en el granero. Su nombre es JAVY. Nadie sabe por qué pudo escapar, y ahora tampoco entiendo cómo lo has conseguido tú. Los *hodis* se retrasaron, o vosotros despertasteis antes de lo que deberíais. O tal vez ambas cosas. También se acercó a nosotros, y se enteró de toda la historia, como estás haciendo tú. Pero él no odiaba los ordenadores porque no los entendiera, sino por lo contrario. Los conocía demasiado bien como para que le gustaran. No le resultó difícil entender este mundo, y se dispuso a destruirlo desde dentro, cacharreando y haciendo cosas que ninguno entendimos. Nos prometió que sería el último en marcharse; que nunca se iría si no lo hacía con todos nosotros. Nosotros juramos ayudarle en lo que necesitara.

— ¿Y qué pasó?

— Algo salió mal. Le dio tiempo a ayudar a algunos de nosotros, que pudieron dejar los campos y unirse a él, apoyándole cuando lo necesitaba. Pero un día los *hodis* le cogieron. Los que habían tenido la suerte de abandonar durante una temporada el campo volvieron rápidamente a la era para evitar represalias.

De repente sientes una curiosidad creciente por aquella historia. No eres aún consciente de que acabas de entrar a formar parte de ella.

— ¿Qué pasó con JAVY? – preguntas.

— Trataron de convencernos de que le habían juzgado limpiamente, pero todos sabemos que no fue así. Le maltrataron, le hicieron hablar, y le encerraron de por vida. Algunos de nosotros tenemos la firme creencia de que con todo lo que sabe, sería capaz de escapar, pero no lo hace, y no entendemos por qué. Quienes creen que no es capaz de fugarse de la prisión lo hacen por decepción. Muchos de ellos son los que un día se unieron a él. Pero ahora has llegado tú, no te han capturado. Eso es un buen principio. ¡¡Tienes que seguir los pasos de JAVY!! Por favor... no sabes lo que es esto...

— ¿¿Pero qué estás diciendo?? Yo no soy él. Yo odio los ordenadores por el lado de la ignorancia y no sabría cómo empezar. No tengo nada que hacer. En algún momento me cogerán, y labraré la tierra junto a vosotros.

— ¡¡¡NO!!! ¡¡Eso no puede ser!! ¡Ahora que has llegado eres nuestra esperanza! ¡Busca a JAVY, convéncele, libérale, y sálvanos a todos!

B.3. JAVY

Esquivando *hodis* llegas a la cárcel y consigues hablar con JAVY. Al principio se asombra mucho de verte, luego te cuenta su propia historia... de la que entiendes sólo algunos trozos.

— El mundo en el que estamos – te dice – es una Máquina Virtual de Java, aunque aquí, por abreviar, se la conoce por sus siglas inglesas, JVM. Sólo el que es capaz de entenderla y manipularla correctamente logrará volver a la realidad.

— Y tú la entiendes...

— Sí. Y algunas otras cosas que el resto de personas encerradas aquí no comprende bien. Cuando llegué, capturé a un *hodi* y le interrogué por la fuerza. No fue difícil. Había muchas preguntas a las que no supo contestarme, su Inteligencia Artificial no era demasiado sofisticada, tal vez sólo un prototipo. Pero sí me contó algunas cosas interesantes. Sentí tentación de matarle al acabar; al fin y al cabo era un simple *hodi*, una instancia de algo inmaterial. Pero no pude. Era un ser autónomo. Vivía... aunque fuera a su manera. Sentí respeto por él, por su "raza", por quien les hubiera creado.

Alguna vez antes habías oído hablar de la educación cibernética, sobre todo en la televisión. Nunca entendiste lo que era, seguramente porque siempre cambiabas inmediatamente de canal, pero no crees que se refirieran a eso. Sentir respeto por una vida artificial. Qué ridiculez.

— No quería matarlo, borrarlo, abortarlo, destruirlo... o cómo diablos pueda decirse. Pero tampoco podía liberarle alegremente y confiar en que no me traicionara. A si es que le borré la RAM. Un modo sencillo de provocar amnesia a uno de esos ... de esos seres.

¿RAM? A ti eso sólo te sonaba a marca de leche.

— Por desgracia, no lo debí hacer del todo bien. Diablos de recolector de basura. A veces falla. Pude comprobarlo después, cuando me cazaron.

— No sé de qué estás hablando. Pero, ¿qué conseguiste sacar al *hodi*?

Hodi... hacía sólo unas cortas horas que habías oído por primera vez esa palabra y ya se había asentado firmemente en el fondo de tu mente.

— Me contó cosas interesantes que no he dicho a nadie, por mi propio beneficio. Si te las cuento, tienes que prometerme que no saldrás de aquí si no es conmigo.

Vaya. Te estaba pidiendo una promesa a cambio de información. Pero dentro de esa información, sospechabas, residía la letra pequeña de tu juramento. Estabas en una encrucijada. Tal vez habría sido mejor decir que no, e intentar capturar un *hodi* tú mismo para sonsacarle la información. Pero lo de "JVM", "RAM" y el recolector de basura te había intimidado. No tenías ni idea de qué era todo aquello, y no tuviste más remedio que aceptar.

— Es muy sencillo – continuó. – El resto de la gente cree que esto es como un castigo, al igual que lo creen los presos que están encerrados en las cárceles de nuestro mundo o, mejor, los adolescentes de los reformatorios. Sin embargo, si preguntas a los responsables, te asegurarán que ellos lo que persiguen es la reinserción. A mí me dijo lo mismo. Estamos aquí encerrados por ser *adigitalis*, por ir en contra de la corriente que impera en la sociedad que nos ha tocado vivir. En eso acertaron los campesinos.

— Sí, eso es lo que me ha contado Wadet.

— Lo que no te contó es lo de la JVM. Lo que pretenden no es encerrarnos sin más. Aquí lo que quieren, al menos esa era la idea inicial, es que nos integremos con un ordenador, que veamos que no es tan horrible, que lo comprendamos desde dentro. Cuando entramos, a cada uno se nos asocia una prueba que demuestre que, al resolverla, hemos aprendido lo suficiente y se nos dejaría salir.

— ¿¿Así de fácil?? Eso no puede ser. ¿Cuál es la prueba de cada uno? ¿Por qué entonces nos convierten en campesinos?

— Te he dicho que esa era la idea inicial. La pers.... el ser al que interrogué no quería continuar, y me costó conseguir que siguiera hablando. Al parecer

algo salió mal en el programa que crea todo este mundo. Tal vez los seres digitales que los controlan han tomado conciencia de su existencia lógica frente a nuestra existencia física y se han vuelto rencorosos. Las ordenes que dan a sus subordinados son ocultar esa prueba: no dejarnos salir jamás.

¿Estaba hablando seriamente de aquello? ¿Cómo podía tener conciencia un *hodi*? Si realmente tenía esa opinión, tal vez no fuera tan descabellada su duda moral a destruir uno de ellos...

— Y ¿cómo podemos saber cual es esa prueba? Y... ¿por qué no superaste la tuya sin más y te marchaste? ¿¿Por qué me has hecho prometer que no me iría si no era contigo??

— Tienes menos anticipación de la que yo creía, amigo. La prueba no es otra cosa que utilizar la JVM para solucionar un problema. Si superas tu problema, te puedes marchar, sin más. Pero mi prueba era la más difícil de todas. No podía resolverla sólo. Yo era capaz de resolver casi todas las pruebas de los demás, pero la mía no. Necesitaba que alguien me ayudara. Busqué una solución, y sólo se me ocurrió ocultar todo esto a los campesinos. Les dije que teníamos que ir resolviendo problemas, y cuando los resolviéramos todos, nos dejarían marchar. La realidad era sólo ligeramente diferente. Con cada problema resuelto, un campesino dejaba de serlo. Yo les oculté que podían irse, y en lugar de eso les dije que al quedar libres del trabajo en el campo, me podían ayudar a salvar más gente en lugar de quedarse mirando, esperando a que yo hiciera todo el trabajo.

— ¡Qué truco más vil! ¿Por qué no les dijiste la verdad?

— Porque no me fío de ellos. Ahí hay gente de todo tipo, Mortando. Algunos me habrían sido leales, pero otros no. Al quedarse junto a mí, yo podía ir dedicándome a problemas cada vez más complicados, mientras ellos me solucionaban el trabajo monótono y repetitivo que ya me habían visto resolver a mí antes.

— ¿Cómo sabes cuales son esas pruebas?

— La JVM lo dice. También dice el nombre de la persona a la que está destinada la prueba. Un determinado problema sólo está activo un tiempo. Si esa persona no lo resuelve, el problema cambia y es el turno de otro campesino. Yo me encargué de que el cartel con el nombre no fuera visible a los demás.

— ¡¡Eres un traidor!! ¡Mereces estar entre estos barrotes!

— No te pongas así. Tú habrías hecho lo mismo. Es más, *vas* a hacer lo mismo.

— ¿¿Yo?? ¡Ni hablar! A ti no te funcionó, como prueba tu situación actual. Por cierto, ¿qué ocurrió?

— La JVM no es tan tonta como parece. Se supone que cada problema está planteado para una persona particular. Sin embargo siempre era yo el que lo resolvía, con la ayuda a veces de los ya libres. Había que hacerse

pasar por el destinatario del problema para que la máquina creyera que era él quien lo estaba resolviendo. Pero cometimos un error y nos pilló. Los *hodis* nos encerraron, tanto a mí como a todos los libres. Los cabecillas resentidos, los que no quieren que intentemos las pruebas para poder marchar, privaron de nuevo a los campesinos de su no utilizada libertad, y a mí, que no había superado mi prueba, me metieron en esta prisión enana para que me pudriera el resto de mi vida.

— ¿Se enteraron los campesinos de que durante un tiempo podrían haber salido de este sitio?

— Afortunadamente no. Gracias a eso, aún nos queda alguna esperanza. Puedes volver a engañarles, como hice yo, para liberarles a todos y también a mí. Recuerda que me has jurado no marcharte sin mí. Eso te obliga a superar todas las pruebas, incluida la mía que, sospecho, será la más difícil de todas.

— No. No puedo hacerlo. No sé nada de cómo funciona esto. ¿¿Por qué no lo has vuelto a intentar tú??

— ¿¿Por qué va a ser?? Por miedo. Me lo hicieron pasar muy mal cuando me cogieron. Intentaron por todos los medios convertirme en un campesino, pero me resistía. Son cabezotas esos seres. Tienen bucles con condiciones difíciles de romper. Y el rencor de un interrogatorio mal olvidado les convierte en seres crueles, te lo aseguro. No quiero volver a pasar por todo aquello si alguien mete la pata y me vuelve a caer a mí la responsabilidad. Tienes que ser tú. Yo siempre estaré en la sombra, dándote consejos y explicándote lo que tienes que hacer cuando lo necesites. Pero no volveré a poner una mano en la JVM que me incrimine. Tú decides. Asumes el riesgo para liberarnos, o vives hasta el fin de tus días escondiéndote para que no te capturen y acabes trabajando de Sol a Sol.

No pareces tener alternativas...

Apéndice C

Formato de los tres tipos de conocimiento de JV^{2}M

C.1. Jerarquía conceptual

C.1.1. DTD

```
<!ELEMENT Ontology (Header, Concept*)>
<!ELEMENT Header (Title, Language, Date, LogSequence?)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT LogSequence (LogEntry*)>
<!ELEMENT LogEntry (#PCDATA)>
<!ATTLIST LogEntry
    date CDATA #REQUIRED
>
<!ELEMENT Concept (Name, Description, ShortDescription?,
                   RelatedTo*, Comment?,
                   SignatureInformation?, Position?)>
<!ATTLIST Concept
    Id ID #REQUIRED
    Type (compilation | instruction | microinstruction |
          structure | metaphor) #REQUIRED
>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT ShortDescription (#PCDATA)>
<!ELEMENT RelatedTo (Sentence*)>
<!ATTLIST RelatedTo
    concept IDREF #REQUIRED
```

```
>
<!ELEMENT Sentence (#PCDATA)>
<!ELEMENT Comment (#PCDATA)>
<!ELEMENT SignatureInformation (Parameter*, Result*)>
<!ELEMENT Parameter EMPTY>
<!ATTLIST Parameter
      Id CDATA #REQUIRED
>
<!ELEMENT Result EMPTY>
<!ATTLIST Result
      Id CDATA #REQUIRED
>
<!ELEMENT Position EMPTY>
<!ATTLIST Position
   x CDATA #REQUIRED
   y CDATA #REQUIRED
>
```

C.1.2. Porción de la jerarquía conceptual

El siguiente código XML representa la porción de la jerarquía conceptual mostrada en la figura 5.5.

```
<Concept Id="bipush" Type="instruction">
    <Name>instrucción load</Name>
    <Description>
        Sirve para introducir en la pila de operandos
        una constante. El valor a apilar se recibe como
        operando de la instrucción.
    </Description>
    <ShortDescription>
        Esta instrucción servía para introducir una
        constante entera en la pila.
    </ShortDescription>
    <SignatureInformation>
        <Parameter>valor a apilar</Parameter>
    </SignatureInformation>
    <Position x="346" y="152"/>
    <RelatedTo concept = "push">
        <Sentence>¿Cómo apilo?</Sentence>
    </RelatedTo>
</Concept>
<Concept Id="push" Type="microinstruction">
    <Name>apilar el valor</Name>
    <Description>
```

```
        Hay que hacer un push. Para eso lleva el
        operando del inventario a la pila de operandos.
    </Description>
    <ShortDescription>
        Esta microinstrucción apila un elemento.
    </ShortDescription>
    <SignatureInformation>
        <Parameter>valor a apilar</Parameter>
    </SignatureInformation>
    <Position x="350" y="248"/>
    <RelatedTo concept = "operandStack">
        <Sentence>¿Qué es la pila de operandos?</Sentence>
    </RelatedTo>
    <RelatedTo concept = "inventory">
        <Sentence>¿Qué es el inventario?</Sentence>
    </RelatedTo>
</Concept>
<Concept Id="operandStack" Type="structure">
    <Name>pila de operandos</Name>
    <Description>
        La pila de operandos es la pila que se utiliza
        para almacenar los resultados intermedios del
        cálculo de expresiones aritméticas complejas.
    </Description>
    <Position x="244" y="352"/>
</Concept>
<Concept Id="inventory" Type="metaphor">
    <Name>inventario</Name>
    <Description>
        Es el lugar donde se depositan los objetos
        intermedios que tienes que manipular para
        ejecutar las instrucciones.
    </Description>
    <Position x="414" y="352"/>
</Concept>
```

C.2. Ejercicios

C.2.1. DTD

```
<!ELEMENT ScenarioBase (Header, Scenario*)>
<!ELEMENT Header (Title, Language, Date, LogSequence)>
<!ELEMENT Title (#PCDATA)>
```

```
<!ELEMENT Language (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT LogSequence (LogEntry)>
<!ELEMENT LogEntry (#PCDATA)>
<!ATTLIST LogEntry
  date CDATA #REQUIRED
>
<!ELEMENT Scenario (Name, Description, ConceptList, Code)>
<!ATTLIST Scenario
  shortExecution (true | false) #REQUIRED
>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT ConceptList (Concept+)>
<!ELEMENT Concept EMPTY>
<!ATTLIST Concept
  id CDATA #REQUIRED
>
<!ELEMENT Code (Class+)>
<!ELEMENT Class (ClassFile, SourceCode, ExplanationTree)>
<!ATTLIST Class
  id CDATA #REQUIRED
>
<!ELEMENT SourceCode (Line*)>
<!ELEMENT Line (#PCDATA)>
<!ELEMENT ExplanationTree (MethodExplanation*)>
<!ELEMENT MethodExplanation (Block*)>
<!ELEMENT Block (Explanation?, RelatedToParent?,
                 RelatedTo?, Block*)>
<!ATTLIST Block
  initLine CDATA #REQUIRED
  initCol CDATA #REQUIRED
  endLine CDATA #REQUIRED
  endCol CDATA #REQUIRED
  initInstr CDATA #REQUIRED
  endInstr CDATA #IMPLIED
>
<!ELEMENT Explanation (#PCDATA)>
<!ATTLIST MethodExplanation
  name CDATA #REQUIRED
>
<!ELEMENT RelatedTo (Sentence)*>
<!ATTLIST RelatedTo
  id CDATA #REQUIRED    <!-- A la jerarquía conceptual -->
```

```
>
<!ELEMENT RelatedToParent (Sentence*)>
<!ELEMENT Sentence (#PCDATA)>
<!ELEMENT ClassFile (InterfaceSequence?,
                     FieldTable?, MethodTable?)>
<!ATTLIST ClassFile
   name CDATA #REQUIRED
   superclass CDATA #IMPLIED
   accessFlag CDATA #REQUIRED
>
<!ELEMENT Interface (#PCDATA)>
<!ELEMENT InterfaceSequence (Interface*)>
<!ELEMENT FieldTable (Field*)>
<!ELEMENT Field EMPTY>
<!ATTLIST Field
   name CDATA #REQUIRED
   type CDATA #REQUIRED
   accessFlag CDATA #REQUIRED
   defaultValue CDATA #IMPLIED
>
<!ELEMENT MethodTable (Method)*>
<!ELEMENT Method (IN*)>
<!ATTLIST Method
   name CDATA #REQUIRED
   type CDATA #REQUIRED
   accessFlag CDATA #REQUIRED
   maxStack CDATA #IMPLIED
   maxLocal CDATA #IMPLIED
>
<!ELEMENT IN (Target*)>
<!ATTLIST IN
   opcode CDATA #REQUIRED
   param CDATA #IMPLIED
   param1 CDATA #IMPLIED
   param2 CDATA #IMPLIED
   className CDATA #IMPLIED
   name CDATA #IMPLIED
   type CDATA #IMPLIED
   value CDATA #IMPLIED
   default CDATA #IMPLIED
   low CDATA #IMPLIED
   high CDATA #IMPLIED
>
<!ELEMENT Target EMPTY>
```

```
<!ATTLIST Target
   offset CDATA #REQUIRED
   value CDATA #IMPLIED
>
```

C.2.2. Porción de un ejercicio

El siguiente código XML representa la porción del ejercicio mostrado en la figura 5.7.

```
<Code>
    <Class id="main">
        <ClassFile name = "main" flags="32">
            <MethodTable>
                <Method name=":init" [...]>
                    <!-- [ ... Constructor ... ] -->
                </Method>
                <Method name="main"
                        type="([Ljava/lang/String;)V"
                        accessFlag="9" maxStack="1"
                        maxLocal="2">
                    <IN opcode="16"
                        textualOpcode="bipush"
                        param="5"/>
                    <IN opcode="16"
                        textualOpcode="bipush"
                        param="3"/>
                    <IN opcode="104"
                        textualOpcode="imul"/>
                    <IN opcode="60"
                        textualOpcode="istore_1"/>
                    <IN opcode="177"
                        textualOpcode="return"/>
                </Method>
            </MethodTable>
        </ClassFile>
        <SourceCode>
            <Line>class main {</Line>
            <Line>  public static void main (
                            String [] args ) {</Line>
            <Line>      int c;</Line>
            <Line>      c = 5*3;</Line>
            <Line>  }</Line>
            <Line>}</Line>
        </SourceCode>
```

```
<ExplanationTree>
    <MethodExplanation name=":init">
        <!-- [ ... Constructor ... ] -->
    </MethodExplanation>
    <MethodExplanation name="main">
        <!-- [ ... Bloques superiores ... ] -->
        <Block initLine="3" initCol="6"
               endLine="3" endCol="9"
               initInstr="0" endInstr="1">
            <!-- Bloque etiquetado con un 3 -->
            <Explanation>
                El primer paso para realizar una
                operación aritmética, en este
                caso una multiplicación, es
                siempre apilar los dos operandos
                en la pila de operandos.
            </Explanation>
            <RelatedTo id="operandStack">
                <Sentence>
                    ¿Qué es la pila de operandos?
                </Sentence>
            </RelatedTo>
            <Block initLine="3" initCol="6"
                   endLine="3" endCol="7"
                   initInstr="0" endInstr="0">
                <Explanation>
                    Para calcular 5*3, lo
                    primero que hay que hacer
                    es apilar el primer
                    operando, es decir, el 5.
                </Explanation>
                <RelatedToParent>
                    <Sentence>
                        ¿Por qué?
                    </Sentence>
                </RelatedToParent>
                <RelatedTo id="loadConst">
                    <Sentence>
                        ¿Qué hace la instrucción
                        actual?
                    </Sentence>
                </RelatedTo>
            </Block>
            <Block initLine="3" initCol="8"
```

```
                                endLine="3"  endCol="9"
                                initInstr="1"  endInstr="1">
                            <Explanation>
                                Ya esta apilado el primer
                                operando (el 5). Ahora hay
                                que apilar el segundo, el 3.
                            </Explanation>
                            <RelatedToParent>
                                <Sentence>
                                    ¿Apilar?
                                </Sentence>
                            </RelatedToParent>
                            <RelatedTo id="loadConst">
                                <Sentence>
                                    ¿Qué hace la instrucción
                                    actual?
                                </Sentence>
                            </RelatedTo>
                        </Block>
                    </Block>
                    <!-- [ ... Bloques siguientes ... ] -->
                </MethodExplanation>
            </ExplanationTree>
        </Class>
</Code>
```

C.3. Grafos de ejecución

C.3.1. DTD

```
<!ELEMENT ExecutionInformation (Header, Graph*)>
<!ELEMENT Header (Title, Language,
                  Date, LogSequence, Templates, Sentences)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT LogSequence (LogEntry)>
<!ELEMENT LogEntry (#PCDATA)>
<!ATTLIST LogEntry date CDATA #REQUIRED >
<!ELEMENT Templates (GoalTemplates?, AgentGoalTemplates?,
                     StudentGoalTemplates?)>
<!ELEMENT GoalTemplates (Template+)>
<!ELEMENT AgentGoalTemplates (Template+)>
```

```
<!ELEMENT StudentGoalTemplates (Template+)>
<!ELEMENT AdviseTemplate (Template+)>
<!ELEMENT Template (#PCDATA)>
<!ELEMENT Sentences (Sentence+)>
<!ELEMENT Sentence (#PCDATA)>
<!ELEMENT Graph (Name, DefaultMnemonic, OpCode*,
                 VariableList?, Operand*, Node*)>
<!ATTLIST Graph
  id ID #REQUIRED
  firstNode CDATA #REQUIRED
>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT DefaultMnemonic (#PCDATA)>
<!ELEMENT OpCode EMPTY>
<!ATTLIST OpCode
  value CDATA #REQUIRED
  mnemonic CDATA #IMPLIED
>
<!ELEMENT VariableList (Var*)>
<!ELEMENT Var EMPTY>
<!ATTLIST Var
  id CDATA #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT Operand EMPTY>
<!ATTLIST Operand
  id CDATA #REQUIRED
  var CDATA #REQUIRED
>
<!ELEMENT Node (Comment?, Edge*, Position?)>
<!ATTLIST Node
  id ID #REQUIRED
  type (state | shortExecution | conditionTrue |
        availableMethod | incPC | noIncPC) #IMPLIED
>
<!ELEMENT Comment (#PCDATA)>
<!ELEMENT Edge (Signature?, Explanation?)>
<!ATTLIST Edge
  microinstruction CDATA #REQUIRED
  valid (false | true) #IMPLIED
  target CDATA #REQUIRED
>
<!ELEMENT Explanation (#PCDATA)>
<!ELEMENT Signature (Param?, Result?)>
```

```
<!ELEMENT Param EMPTY>
<!ATTLIST Param
   id CDATA #REQUIRED
   var CDATA #REQUIRED
>
<!ELEMENT Result EMPTY>
<!ATTLIST Result
   id CDATA #REQUIRED
   var CDATA #REQUIRED
>
<!ELEMENT Position EMPTY>
<!ATTLIST Position
   x CDATA #REQUIRED
   y CDATA #REQUIRED
>
```

C.3.2. Ejemplo de un grafo de ejecución

El siguiente código XML representa el grafo de ejecución mostrado en la figura 5.10.

```
<ExecutionInformation>
    <!-- [ ... Otras cosas ... ] -->
    <Header>
        <Templates>
            <GoalTemplates>
                <Template>Para %o hay que %m</Template>
                <Template>Se debe %m para %o</Template>
                <Template>Hay que %m para %o</Template>
            </GoalTemplates>
            <AgentGoalTemplates>
                <Template>Para %o tengo que %m</Template>
                <Template>Voy a %m para así %o</Template>
                <Template>Tengo que %m para %o</Template>
            </AgentGoalTemplates>
            <AdviseTemplates>
                <Template>
                    Si yo fuera tú intentaría %o
                </Template>
                <Template>
                    Deberías tratar de %o
                </Template>
            </AdviseTemplates>
            <StudentGoalTemplates>
                <Template>
```

```
                Si quieres %o deberías tratar de %m
            </Template>
            <Template>
                Como quieres %o deberías %m
            </Template>
            <Template>
                Para %o podrías probar a %m
            </Template>
        </StudentGoalTemplates>
    </Templates>
    <Sentences>
        <Sentence>
            Cuentame algo más sobre eso de %m
        </Sentence>
        <Sentence>
            ¿Y qué es eso de '%m' exactamente?
        </Sentence>
    </Sentences>
</Header>
<Graph firstNode="Begin 1" id="bipush">
    <VariableList>
        <Var id="val" name="valor a apilar"/>
    </VariableList>
    <Operand id="constant" var="val"/>
    <Node id="Begin 1" type="state">
        <Edge microinstruction="push"
              target="End 1" valid="true">
            <Signature>
                <Param id="value" var="val"/>
            </Signature>
            <Explanation>
                introducir en la pila el operando
                de la instrucción
            </Explanation>
        </Edge>
        <Edge microinstruction="pushFrame"
              target="node1" valid="false">
            <Signature/>
            <Explanation>
                No, no. Ese es Framauro que te
                ayudará en otro momento.
            </Explanation>
        </Edge>
        <Position x="304" y="96"/>
```

```
            </Node>
            <Node id="End 1" type="noIncPC">
                <Position x="309" y="344"/>
            </Node>
            <Node id="node1" type="state">
                <Position x="504" y="208"/>
            </Node>
        </Graph>
        <!-- [ ... otros grafos ... ] -->
</ExecutionInformation>
```

Apéndice D

Ontología de Java

```
# Base: http://gaia.fdi.ucm.es/java#
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix default: <http://gaia.fdi.ucm.es/java#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

default:ReferenceType
  a owl:Class ;
  owl:equivalentClass
    [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Class ;
            owl:unionOf (
              default:Array
              default:ClassOrInterface)
        ] default:Type)
    ] ;

default:int
  a default:IntegralType .

default:isFollowedBy
  a owl:TransitiveProperty , owl:ObjectProperty ;
  rdfs:subPropertyOf default:hasListProperty .

default:IntegralType
  a owl:Class ;
  owl:disjointWith default:FloatingPointType ;
  owl:equivalentClass
```

```
    [ a owl:Class ;
      owl:intersectionOf ([ a owl:Class ;
      owl:oneOf (default:byte default:short
                 default:int default:long default:char)
                         ] default:NumericType)
    ] .

default:implements
  a owl:ObjectProperty ;
  rdfs:domain default:Class ;
  rdfs:range default:Interface ;
  owl:inverseOf default:implementedBy .

default:StringLiteral
  a owl:Class ;
  owl:disjointWith default:FloatingPointLiteral ,
                   default:BooleanLiteral ,
                   default:CharacterLiteral ,
                   default:IntegerLiteral ;
  owl:equivalentClass
    [ a owl:Class ;
      owl:intersectionOf (default:Literal [
      a owl:Restriction ;
      owl:cardinality "1"^^xsd:int ;
      owl:onProperty default:hasStringValue
                                          ])
    ] .

default:hasFirstOperand
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:BinaryOperatorExpression ;
  rdfs:range default:Expression .

default:Block
  a owl:Class ;
  rdfs:subClassOf owl:Thing ;
  rdfs:subClassOf
    [ a owl:Class ;
      owl:unionOf ([ a owl:Restriction ;
      owl:onProperty default:hasContents ;
      owl:someValuesFrom default:BlockStatement
      ]
      [ a owl:Restriction ;
          owl:allValuesFrom
```

```
        [ a  owl:Class ;
             owl:intersectionOf
          (default:Block
            [ a owl:Restriction ;
                owl:onProperty
                  default:hasContents ;
                owl:someValuesFrom
                  default:BlockStatement
            ])
        ] ;
        owl:onProperty default:hasNext
      ])
    ] ;
  owl:disjointWith default:ArgumentList ,
                   default:BlockStatement ,
                   default:Package ,
                   default:Initializer ,
                   default:SourceFile ,
                   default:FormalParameterList .

default:char
  a default:IntegralType .

default:hasLiteral
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:range default:Literal .

default:extendedBy
  a owl:ObjectProperty ;
  rdfs:domain default:NonFinalClass ;
  rdfs:range default:Class ;
  owl:inverseOf default:extends .

default:hasStringValue
  a owl:DatatypeProperty ;
  rdfs:range xsd:string ;
  rdfs:subPropertyOf default:hasValue .

default:isDeclaredIn
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Method ;
  rdfs:range default:ClassOrInterface ;
  owl:inverseOf default:hasMethod .
```

```
default:double
  a default:FloatingPointType .

default:NamedElement
  a owl:Class ;
  owl:equivalentClass
    [ a owl:Restriction ;
        owl:cardinality "1"^^xsd:int ;
        owl:onProperty default:hasName
    ] .

default:ArgumentList
  a owl:Class ;
  rdfs:subClassOf owl:Thing ;
  rdfs:subClassOf
  [ a owl:Class ;
      owl:unionOf (
        [ a owl:Restriction ;
            owl:onProperty default:hasContents ;
            owl:someValuesFrom default:Expression
        ]
        [ a owl:Restriction ;
            owl:allValuesFrom
            [ a owl:Class ;
              owl:intersectionOf (
                default:ArgumentList
                  [ a owl:Restriction ;
                      owl:onProperty
                      default:hasContents ;
                      owl:someValuesFrom
                      default:Expression
                  ])
            ] ;
            owl:onProperty default:hasNext
        ])
  ] ;
  owl:disjointWith default:BlockStatement ,
                   default:Block ,
                   default:Package ,
                   default:SourceFile ,
                   default:Initializer ,
                   default:FormalParameterList .

default:hasSecondOperand
```

```
  a owl:FunctionalProperty ,
    owl:ObjectProperty ;
  rdfs:domain default:BinaryOperatorExpression ;
  rdfs:range default:Expression .

default:hasName
  a owl:FunctionalProperty ,
    owl:DatatypeProperty ;
  rdfs:domain default:NamedElement ;
  rdfs:range xsd:string .

default:hasDeclarationOfVariable
  a owl:FunctionalProperty ,
    owl:ObjectProperty ;
  rdfs:domain default:Declarator ;
  rdfs:range default:Variable .

default:IntegerLiteral
  a owl:Class ;
  owl:disjointWith default:FloatingPointLiteral ,
                   default:BooleanLiteral ,
                   default:CharacterLiteral ,
                   default:StringLiteral ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf
        (default:Literal
          [ a owl:Restriction ;
              owl:cardinality "1"^^xsd:int ;
              owl:onProperty
                default:hasIntegerValue
          ]
        )
  ] .

default:isSuperClassOf
  a owl:TransitiveProperty , owl:ObjectProperty ;
  rdfs:domain default:NonFinalClass ;
  rdfs:range default:Class ;
  owl:inverseOf default:isSubclassOf .

default:hasIntegralTypeEnum
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:range default:IntegralType .
```

```
default:hasVisibility
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:range default:Visibility .

default:Declarator
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Restriction ;
            owl:onProperty
              default:hasDeclarationOfVariable ;
            owl:someValuesFrom default:Variable
        ] default:BlockStatement)
  ] .

default:hasDeclarationOfClassOrInterface
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:SourceFile ;
  rdfs:range default:ClassOrInterface .

default:isInitializedBy
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Declarator ;
  rdfs:range default:Expression .

default:FloatingPointLiteral
  a owl:Class ;
  owl:disjointWith default:BooleanLiteral ,
                   default:IntegerLiteral ,
                   default:CharacterLiteral ,
                   default:StringLiteral ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Restriction ;
            owl:cardinality "1"^^xsd:int ;
            owl:onProperty
              default:hasFloatingPointValue
        ] default:Literal)
  ] .

default:private
```

```
  a default:Visibility .

default:hasFloatingPointTypeEnum
  a owl:FunctionalProperty ,
    owl:ObjectProperty ;
  rdfs:range default:FloatingPointType .

default:public
  a default:Visibility .

default:FloatingPointType
  a owl:Class ;
  owl:disjointWith default:IntegralType ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Class ;
            owl:oneOf (
              default:float
              default:double
            )
        ] default:NumericType)
  ] .

default:ClassOrInterface
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        default:ReferenceType
        [ a owl:Class ;
            owl:unionOf
              (default:Class default:Interface)
        ])
  ] .

default:EnhancedFor
  a owl:Class ;
  rdfs:subClassOf default:For ;
  owl:disjointWith default:BasicFor .

default:hasForUpdate
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:For ;
```

```
  rdfs:range default:Expression .

[] a owl:AllDifferent ;
  owl:distinctMembers
    (default:byte default:char default:double
     default:float default:int default:long
     default:null default:private
    default:protected default:public
    default:short) .

default:Expression
  a owl:Class ;
  rdfs:subClassOf default:Initializer .

default:Variable
  a owl:Class ;
  rdfs:subClassOf default:TypedElement ,
                  default:Terminal ,
                  default:NamedElement ;
  owl:disjointWith default:Package .

default:Constructor
  a owl:Class ;
  rdfs:subClassOf default:InstanceMethod ;
  rdfs:subClassOf
  [ a owl:Restriction ;
      owl:cardinality "0"^^xsd:int ;
      owl:onProperty default:hasReturnType
  ] .

default:CharacterLiteral
  a owl:Class ;
  owl:disjointWith
    default:FloatingPointLiteral ,
    default:BooleanLiteral ,
    default:IntegerLiteral ,
    default:StringLiteral ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Restriction ;
            owl:cardinality "1"^^xsd:int ;
            owl:onProperty default:hasCharacterValue
        ] default:Literal)
```

```
  ] .

default:float
  a default:FloatingPointType .

default:hasCharacterValue
  a owl:DatatypeProperty ;
  rdfs:range xsd:string ;
  rdfs:subPropertyOf default:hasValue .

default:AbstractMethod
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Restriction ;
            owl:cardinality "0"^^xsd:int ;
            owl:onProperty default:hasBody
        ] default:InstanceMethod)
  ] .

default:FinalClass
  a owl:Class ;
  rdfs:subClassOf default:Class ;
  owl:disjointWith default:NonFinalClass .

default:hasForTermination
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:For ;
  rdfs:range default:Expression .

default:isInPackage
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:ClassOrInterface ;
  rdfs:range default:Package ;
  owl:inverseOf
    default:hasClassOrInterfaceDeclaration .

default:MethodCall
  a owl:Class ;
  rdfs:subClassOf default:BlockStatement ;
  rdfs:subClassOf
  [ a owl:Restriction ;
      owl:onProperty default:hasMethod ;
```

```
      owl:someValuesFrom default:Method
  ] ;
  owl:disjointWith default:LoopStatement .

default:hasImport
  a owl:ObjectProperty ;
  rdfs:domain default:SourceFile ;
  rdfs:range default:Package .

default:hasValue
  a owl:FunctionalProperty ,
    owl:DatatypeProperty .

default:LoopStatement
  a owl:Class ;
  rdfs:subClassOf default:BlockStatement ;
  owl:disjointWith default:MethodCall .

default:Interface
  a owl:Class ;
  rdfs:subClassOf default:ClassOrInterface ;
  rdfs:subClassOf
  [ a owl:Restriction ;
      owl:allValuesFrom default:AbstractMethod ;
      owl:onProperty default:hasMethod
  ] ;
  owl:disjointWith default:Class .

default:hasContents
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:subPropertyOf default:hasListProperty .

default:byte
  a default:IntegralType .

default:hasForInitialization
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:For ;
  rdfs:range default:Declarator .

default:Array
  a owl:Class ;
  rdfs:subClassOf default:ReferenceType ;
  rdfs:subClassOf
```

```
  [ a owl:Restriction ;
      owl:onProperty default:hasType ;
      owl:someValuesFrom default:Type
  ] .

default:BasicFor
  a owl:Class ;
  rdfs:subClassOf default:For ;
  owl:disjointWith default:EnhancedFor .

default:InstanceMethod
  a owl:Class ;
  rdfs:subClassOf default:Method ;
  owl:disjointWith default:StaticMethod .

default:long
  a default:IntegralType .

default:protected
  a default:Visibility .

default:hasBlock
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:LoopStatement ;
  rdfs:range default:Block .

default:Initializer
  a owl:Class ;
  owl:disjointWith
    default:ArgumentList ,
    default:BlockStatement , default:Block ,
    default:Package , default:SourceFile ,
    default:FormalParameterList .

default:SourceFile
  a owl:Class ;
  rdfs:subClassOf owl:Thing ;
  rdfs:subClassOf
  [ a owl:Restriction ;
      owl:onProperty
        default:hasDeclarationOfClassOrInterface ;
      owl:someValuesFrom default:ClassOrInterface
  ] ;
  owl:disjointWith
```

```
    default:ArgumentList , default:BlockStatement ,
    default:Block , default:Package ,
    default:Initializer ,
    default:FormalParameterList .

<http://gaia.fdi.ucm.es/java>
  a owl:Ontology .

default:OperatorExpression
  a owl:Class ;
  rdfs:subClassOf default:Expression ;
  owl:disjointWith default:Terminal .

default:NonFinalClass
  a owl:Class ;
  rdfs:subClassOf default:Class ;
  owl:disjointWith default:FinalClass .

default:hasType
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:range default:Type .

default:hasArgumentList
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:MethodCall ;
  rdfs:range default:ArgumentList .

default:BinaryOperatorExpression
  a owl:Class ;
  rdfs:subClassOf default:OperatorExpression .

default:implementedBy
  a owl:ObjectProperty ;
  rdfs:domain default:Interface ;
  rdfs:range default:Class ;
  owl:inverseOf default:implements .

default:ClassOrInterfaceInPackage
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Restriction ;
            owl:onProperty default:isInPackage ;
```

```
            owl:someValuesFrom default:Package
        ] default:ClassOrInterface
      )
  ] .

default:invokes
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:MethodCall ;
  rdfs:range default:Method .

default:StaticMethod
  a owl:Class ;
  rdfs:subClassOf default:Method ;
  owl:disjointWith default:InstanceMethod .

default:hasMethod
  a owl:InverseFunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:ClassOrInterface ;
  rdfs:range default:Method ;
  owl:inverseOf default:isDeclaredIn .

default:hasBooleanValue
  a owl:DatatypeProperty ;
  rdfs:range xsd:boolean ;
  rdfs:subPropertyOf default:hasValue .

default:Method
  a owl:Class ;
  rdfs:subClassOf default:NamedElement ;
  rdfs:subClassOf
  [ a owl:Restriction ;
      owl:onProperty default:isDeclaredIn ;
      owl:someValuesFrom default:ClassOrInterface
  ] ;
  rdfs:subClassOf
  [ a owl:Restriction ;
      owl:onProperty default:hasVisibility ;
      owl:someValuesFrom default:Visibility
  ] ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:unionOf (
        default:InstanceMethod
        default:StaticMethod
```

```
      )
  ] .

default:isSubclassOf
  a owl:TransitiveProperty , owl:ObjectProperty ;
  rdfs:domain default:Class ;
  rdfs:range default:NonFinalClass ;
  owl:inverseOf default:isSuperClassOf .

default:short
  a       default:IntegralType .

default:Visibility
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:oneOf (
        default:public
        default:protected
        default:private)
  ] .

default:hasRightHandSide
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Assignment ;
  rdfs:range default:Expression .

default:hasCondition
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:While ;
  rdfs:range default:Expression .

default:BlockStatement
  a owl:Class ;
  owl:disjointWith
    default:ArgumentList , default:Block ,
    default:Package , default:SourceFile ,
    default:Initializer ,
    default:FormalParameterList .

default:Package
  a owl:Class ;
  rdfs:subClassOf owl:Thing ,
    default:NamedElement ;
```

```
  owl:disjointWith
    default:ArgumentList ,
    default:BlockStatement , default:Block ,
    default:Initializer , default:Variable ,
    default:SourceFile ,
    default:FormalParameterList .

default:Type
  a owl:Class ;
    owl:equivalentClass
      [ a owl:Class ;
          owl:unionOf (
            default:PrimitiveType
            default:ReferenceType
          )
      ] .

default:hasClassOrInterfaceDeclaration
  a owl:InverseFunctionalProperty ,
    owl:ObjectProperty ;
  rdfs:domain default:Package ;
  rdfs:range default:ClassOrInterface ;
  owl:inverseOf default:isInPackage .

default:hasReturnType
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Method ;
  rdfs:range default:Type .

default:hasNext
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:subPropertyOf default:isFollowedBy .

default:Literal
  a owl:Class ;
  rdfs:subClassOf default:Terminal ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:unionOf (
        default:IntegerLiteral
        default:FloatingPointLiteral
        default:BooleanLiteral
        default:CharacterLiteral
        default:StringLiteral
```

```
          [ a owl:Class ;
              owl:oneOf (default:null)
          ]
        )
  ] .

default:hasParameterList
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Method ;
  rdfs:range default:FormalParameterList .

default:Class
  a owl:Class ;
  rdfs:subClassOf default:ClassOrInterface ;
  owl:disjointWith default:Interface ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:unionOf (
        default:FinalClass
        default:NonFinalClass
      )
  ] .

default:Addition
  a owl:Class ;
  rdfs:subClassOf default:BinaryOperatorExpression .

default:Terminal
  a owl:Class ;
  rdfs:subClassOf default:Expression ;
  owl:disjointWith default:OperatorExpression .

default:PrimitiveType
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Class ;
            owl:unionOf (
              default:NumericType
                [ a owl:Class ;
                    owl:oneOf (default:bool)
                ]
            )
```

```
        ] default:Type)
  ] .

default:hasBody
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Method ;
  rdfs:range default:Block .

default:hasAbstractMethod
  a owl:ObjectProperty ;
  rdfs:range default:AbstractMethod ;
  rdfs:subPropertyOf default:hasMethod .

default:hasLeftHandSide
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Assignment ;
  rdfs:range default:Variable .

default:NumericType
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Class ;
            owl:unionOf (
              default:IntegralType
              default:FloatingPointType
            )
        ]
        default:PrimitiveType
      )
  ] .

default:hasFloatingPointValue
  a owl:DatatypeProperty ;
  rdfs:range xsd:float ;
  rdfs:subPropertyOf default:hasValue .

default:hasListProperty
  a owl:ObjectProperty .

default:extends
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Class ;
```

```
  rdfs:range default:NonFinalClass ;
  owl:inverseOf default:extendedBy .

default:FormalParameterList
  a owl:Class ;
  rdfs:subClassOf owl:Thing ;
  rdfs:subClassOf
    [ a owl:Class ;
        owl:unionOf (
          [ a owl:Restriction ;
              owl:onProperty default:hasContents ;
              owl:someValuesFrom default:Type
          ]
          [ a owl:Restriction ;
              owl:allValuesFrom
                [ a owl:Class ;
                    owl:intersectionOf (
                      default:FormalParameterList
                      [ a owl:Restriction ;
                          owl:onProperty
                            default:hasContents ;
                          owl:someValuesFrom
                            default:Type
                      ]
                    )
                ] ;
              owl:onProperty default:hasNext
          ]
        )
  ] ;
  owl:disjointWith
    default:ArgumentList , default:BlockStatement ,
    default:Block , default:Package ,
    default:Initializer , default:SourceFile .

default:null
  a default:Literal .

default:hasVariable
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain default:Declarator ;
  rdfs:range default:Variable .

default:hasInitializer
```

```
  a owl:ObjectProperty .

default:Assignment
  a       owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
      owl:intersectionOf (
        [ a owl:Restriction ;
            owl:onProperty default:hasLeftHandSide ;
            owl:someValuesFrom default:Variable
        ]
        [ a owl:Restriction ;
            owl:onProperty default:hasRightHandSide ;
            owl:someValuesFrom default:Expression
        ] default:BlockStatement
      )
  ] .

default:hasIntegerValue
  a owl:DatatypeProperty ;
  rdfs:range xsd:int ;
  rdfs:subPropertyOf default:hasValue .

default:BooleanLiteral
  a owl:Class ;
  owl:disjointWith
    default:FloatingPointLiteral ,
    default:IntegerLiteral ,
    default:CharacterLiteral ,
    default:StringLiteral ;
  owl:equivalentClass
    [ a owl:Class ;
        owl:intersectionOf (
          [ a owl:Restriction ;
              owl:cardinality "1"^^xsd:int ;
              owl:onProperty
                default:hasBooleanValue
          ] default:Literal)
    ] .

default:bool
  a default:PrimitiveType .

default:hasNumericType
```

```
  a owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:range default:NumericType .

default:While
  a owl:Class ;
  rdfs:subClassOf default:LoopStatement ;
  owl:disjointWith default:For .

default:TypedElement
  a owl:Class ;
  owl:equivalentClass
    [ a owl:Restriction ;
        owl:onProperty default:hasType ;
        owl:someValuesFrom default:Type
    ] .

default:For
  a owl:Class ;
  rdfs:subClassOf default:LoopStatement ;
  owl:disjointWith default:While .
```

Bibliografía

Y así, del mucho leer y del poco dormir, se le secó el celebro de manera que vino a perder el juicio.

Don Quijote de la Mancha, Miguel de Cervantes

Aamodt, A. Knowledge-intensive case-based reasoning and sustained learning. En *European Conference on Artificial Intelligence*, páginas 1–6. 1990.

Aamodt, A. y Plaza, E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, vol. 7(1), páginas 39–59, 1994. ISSN 0921-7126.

Advanced Distributed Learning (ADL), editor. *Sharable Content Object Reference Model (SCORM) 2004, 2nd Edition, Overview*. ADL Co-Laboratory, 2004.

Aldrich, C. *Learning by doing, A comprehensive guide to simulations, computer games, and Pedagogy in e-Learing and Other Educational Experiences*. Pfeiffer, 2005. ISBN 978-0-7879-9735-1.

Alonso, C. M. y Gallego, D. J. *Aprendizaje y ordenador*. Dykinson, 2000. ISBN 84-8155-687-4.

Amory, A., Naicker, K., Vincent, J. y Adams, C. The use of computer games as an educational tool: identification of appropriate game types and game elements. *British Journal of Educational Technology*, vol. 30(4), páginas 311–321, 1999.

Anderson, K. Computer-assisted instruction: An overview. *Journal of Medical Systems*, vol. 10(2), páginas 163–171, 1986. ISSN 0148-5598.

Apperley, T. H. Genre and game studies: Toward a critical approach to video game genres. *Simulation & Gaming*, vol. 37(1), páginas 6–23, 2006.

Apps, V. *40 juegos educativos (para Amstrad CPC 464)*. Indescomp, 1985. ISBN 84-86-17620-4.

Aylett, R. S. Emergent narrative, social immersion and "storification". En *Proceedings of Narrative Interaction for Learning Environments*. Edinburgh, 2000.

Ball, G., Ling, D., Kurlander, D., Miller, J., Pugh, D., Skelly, T., Stankosky, A., Thiel, D., Dantzich, M. V. y Wax, T. *Software Agents*, capítulo Lifelike computer characters: The Persona project at Microsoft Research. The MIT Press, 1997. ISBN 978-0-262-52234-2.

Beck, J., Stern, M. y Haugsjaa, E. Applications of AI in education. *Crossroads*, vol. 3(1), páginas 11–15, 1996.

Beck, J. C. y Wade, M. *Got Game: How the Gamer Generation Is Reshaping Business Forever*. Harvard Business School Press, 2004. ISBN 1-57851-949-7.

Bello Tomás, J. J., González Calero, P. A. y Díaz Agudo, B. JColibri: an object-oriented framework for building CBR systems". En *Advances in Case-Based Reasoning, Procs. of the 7th European Conference on Case-Based Reasoning, ECCBR 2004* (editado por P. Funk y P. A. González-Calero), vol. 3155 de *Lecture Notes in Artificial Intelligence, subseries of LNCS*, páginas 32–46. Springer, 2004.

Bennett, F. *Computers as Tutors: Solving the Crisis in Education*. Faben, primera edición, 1999. ISBN 0-9669583-6-5.

Berenguer, X. Historias por ordenador. `http://www.iua.upf.es/~berenguer/textos/histor/narrc.htm`, 1998.

Blanchard, K. y Cheska, A. *The anthropology of sport: An introduction*. Massachusetts: Bergin & Garvey Publisher, Inc., 1985. ISBN 0-89789-041-8.

Bloom, B. S. The 2 Sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, vol. 13(6), páginas 4–16, 1984.

Bopp, M. Didactic analysis of digital games and game-based learning. En *Affective and Emotional Aspects of Human-Computer Interaction: Game-based And Innovative Learning Approaches* (editado por M. Pivec), vol. 1 de *The Future of Learning*, páginas 8–37. IOS Press, 2006. ISBN 1-58603-572-X.

Bork, A. *Learning with computers*. Bedford Mass, 1981. ISBN 0-932376-11-8.

Bowman, R. F. A 'Pac-Man' theory of motivation: tactical implications for classroom instruction. *Educational Technology*, vol. 22(9), páginas 14–17, 1982.

Bradshaw, J. M. An introduction to software agents. En *Software Agents* (editado por J. M. Bradshaw), páginas 3–46. AAAI Press / The MIT Press, 1997.

Brown, E. *That's Edutainment: A Parent's Guide to Educational Software*. Mcgraw-Hill, 1995. ISBN 0-07-882083-9.

Bruner, J. *Toward a Theory of Instruction*. W. W. Norton, 1966. Traducción al español: *Hacia una teoría de la instrucción*, Utema, Méjico, 1969.

Bruno, J. E. Doing time – killing time at school: An examination of the perceptions and allocations of time among teacher-defined at-risk students. *Urban Review*, vol. 27(2), páginas 101–120, 1995. ISSN 0042-0972.

Buchanan, B. G. y Shortliffe, E. H., editores. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley Series in Artificial Intelligence. Addison-Wesley, 1984. ISBN 0-201-10172-6.

Burton, R. R. y Brown, J. S. A tutoring and student modelling paradigm for gaming environments. En *Proceedings of the ACM SIGCSE-SIGCUE technical symposium on Computer science and education*, páginas 236–246. ACM Press, New York, NY, USA, 1976.

Bush, V. As we may think. *The Atlantic Monthly*, vol. 176(1), páginas 101–108, 1945.

Cailliau, R. y Gillies, J. *How the Web Was Born: The Story of the World Wide Web*. Oxford University Press, 2000. ISBN 0-19-286207-3.

Carbonell, J. R. AI in CAI: An Artificial-Intelligence approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems*, vol. 11(4), páginas 190–202, 1970. ISSN 0536-1540.

Card, O. S. *El juego de Ender*. Ediciones B, 1985. ISBN 8466616896.

Carreño de la Cruz, L. y Merino Malillos, L. Videojuegos: Juego y alfabetización digital. En *Claves de la Alfabetización Digital* (editado por E. Díez Calurano), páginas 349–359. Editorial Ariel, 2006.

Cassell, J. y Jenkins, H., editores. *From Barbie to Mortal Kombat: Gender and Computer Games*. The MIT Press, 1998. ISBN 0-262-03258-9.

Castillejo Brull, J. L., Vázquez Gómez, G., Colom Cañellas, A. J. y Sarramona López, J. *Teoría de la educación*. Taurus universitaria, 1993. ISBN 84-306-0088-5.

Clancey, W. J. *Knowledge-based tutoring: the GUIDON program*. MIT Press, 1987. ISBN 0-262-03123-X.

Conati, C., Gertner, A. y Vanlehn, K. Using bayesian networks to manage uncertainly in student modeling. *User Modeling & User-Adapted Interaction*, vol. 12(4), páginas 371–417, 2002. ISSN 0924-1868.

Conklin, J. Hypertext: An introduction and survey. *IEEE Computer*, vol. 20(9), páginas 17–41, 1987.

Coombs, P. H. y Ahmed, M. *Attacking Rural Poverty: How Nonformal Education Can Help. A Research Report for the World Bank Prepared by the International Council for Educational Development*. Johns Hopkins University Press, 1974. ISBN 0-8018-1601-7. Traducción al español: *La lucha contra la pobreza rural. El aporte de la educación no formal.* Publicado para el Banco Mundial por Editorial Tecnos, Madrid, 1975.

Cooper, G. y Sweller, J. Effects of schema acquisition and rule automation on mathematical problem-solving transfer. *Journal of Educational Psychology*, vol. 79(4), páginas 347–362, 1987.

Crawford, C. *The Art of Computer Game Design*. Wolf & Peron, 1982. ISBN 0-88134-117-7. Disponible en `http://www.mindsim.com/MindSim/Corporate/artCGD.pdf`.

Crawford, C. *on Game Design*. Peachpit Press, 2003. ISBN 0-13-146099-4.

Dede, C., Salzman, M. y Loftin, B. Sciencespace: Virtual realities for learning complex and abstract scientific concepts. En *Proceedings IEEE Virtual Reality Annual International Symposium (VRAIS'96)*, páginas 246–253. 1996.

Deubel, P. Selecting curriculum-based software. *Learning & Leading with Technology*, vol. 29(5), páginas 10–16, 2002.

Díaz Agudo, B., Gómez Martín, M. A., Gómez Martín, P. P. y González Calero, P. A. Formal concept analysis for knowledge refinement in case based reasoning. En *AI-2005, the XXV SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence* (editado por M. Bramer, F. Coenen y T. Allen), páginas 233–245. Springer, Cambridge, UK, 2005. ISBN 978-1-84628-225-6.

Díez Calurano, E., editor. *Claves de la Alfabetización Digital*. Editorial Ariel, 2006. Recoge las aportaciones realizadas en las I Jornadas sobre Alfabetización Digital celebradas en Madrid en Febrero de 2.006.

Edmark, K. Unemployment and crime: Is there a connection? *The Scandinavian Journal of Economics*, vol. 107(2), páginas 353–373, 2005.

Elliott, C., Rickel, J. y Lester, J. C. Lifelike pedagogical agents and affective computing: An exploratory synthesis. En *Artifical Intelligence Today* (editado por M. J. Wooldridge y M. Veloso), vol. 1600 de *Lecture Notes in Computer Science*, páginas 195–211. Springer-Verlag, 1999. ISBN 978-3-540-66428-4.

Elliott, J., Adams, L. y Bruckman, A. No magic bullet: 3D video games in education. En *Proceedings of the International Conference of the Learning Sciences (ICLS'02)*. 2002.

Ellis, G. J. Youth in the electronic environment: An introduction. *Youth and Society*, vol. 15(1), páginas 3–12, 1983.

Emihovich, C. y Miller, G. E. Effects of logo and CAI on black first graders' achievement, reflectivity, and self-esteem. *Elementary School Journal*, vol. 88(5), páginas 473–487, 1988.

Feifer, R. G. Sherlock: An intelligent tutoring system for teaching people how to learn. En *Artificial Intelligence and Intelligent Tutoring Systems* (editado por D. Kopec y R. B. Thompson), capítulo 6, páginas 111–127. Ellis Horwood, England, 1992. ISBN 0-13-048430-X.

Fletcher, J. D. Modeling of learner in Computer-Based Instruction. *Journal of Computer-Based Instruction*, vol. 1, páginas 118–126, 1975.

Franklin, S. y Graesser, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. En *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)*, vol. 1193 de *Lecture Notes in Computer Science*, páginas 21–35. Springer-Verlag, 1996. ISBN 3-540-62507-0.

Frasson, C. y Aimeur, E. A comparison of three learning strategies in Intelligent Tutoring Systems. *Journal of Educational Computing Research*, vol. 14(4), páginas 371–383, 1996. ISSN 0735-6331.

Freedman, R., Ali, S. S. y McRoy, S. What is an Intelligent Tutoring System? *Intelligence*, vol. 11(3), páginas 15–16, 2000. ISSN 1523-8822.

Gee, J. P. *What video games have to teach us about learning and literacy*. Palgrave Macmillan, 2004. ISBN 1-4039-6538-2.

Gertner, A. S. y VanLehn, K. Andes: A coached problem solving environment for physics. En *Proceedings of the 5th International Conference on Intelligent Tutoring Systems (ITS 2000)* (editado por G. Gauthier, C. Frasson y K. VanLehn), vol. 1839 de *Lecture Notes in Computer Science*, páginas 133–142. Springer-Verlag, 2000. ISBN 3-540-67655-4.

Gómez Martín, M. A. *Arquitectura y metodología para el desarrollo de sistemas educativos basados en videojuegos*. Tesis Doctoral, Facultad de Informática, Universidad Complutense de Madrid, 2007.

Gómez Martín, M. A., Gómez Martín, P. P. y González Calero, P. A. Aprendizaje basado en juegos. *La Revista Icono 14*, (4), 2004a. ISSN 1697-8293.

Gómez Martín, M. A., Gómez Martín, P. P. y González Calero, P. A. Game-driven Intelligent Tutoring Systems. En *Entertainment Computing - ICEC 2004, Third International Conference, Eindhoven, The Netherlands* (editado por M. Rauterberg), vol. 3166 de *Lecture Notes in Computer Science*, páginas 108–113. Springer, 2004b. ISBN 3-540-22947-7.

Gómez Martín, M. A., Gómez Martín, P. P. y González Calero, P. A. Game-based learning as a new domain for case-based reasoning. En *1st Workshop on Computer Gaming and Simulation Environments, at 6th International Conference on Case-Based Reasoning (ICCBR)* (editado por D. W. Aha y D. Wilson), páginas 175–184. Chicago, IL, US, 2005a.

Gómez Martín, M. A., Gómez Martín, P. P. y González Calero, P. A. Dynamic binding is the name of the game. En *Entertainment Computing (ICEC 2006), 5th International Conference* (editado por R. Harper, M. Rauterberg y M. Combetto), vol. 4161 de *Lecture Notes in Computer Science*, páginas 229–232. Springer, 2006a. ISBN 978-3-540-45259-1. ISSN 0302-9743.

Gómez Martín, M. A., Gómez Martín, P. P. y González Calero, P. A. Aprendizaje activo en simulaciones interactivas. *Revista Iberoamericana de Inteligencia Artificial*, vol. 11(33), páginas 25–36, 2007a.

Gómez Martín, M. A., Gómez Martín, P. P., González Calero, P. A. y Díaz Agudo, B. Adjusting game difficulty level through formal concept analysis. En *AI-2006, the XXVI SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence* (editado por M. Bramer, F. Coenen y T. Allen), páginas 217–230. Springer, Cambridge, UK, 2006b. ISBN 978-1-84628-662-9.

Gómez Martín, M. A., Gómez Martín, P. P., González Calero, P. A. y Jiménez Díaz, G. JV^{2}M, un sistema de enseñanza de la compilación de Java. En *I Simposio Nacional de Tecnologías de la Información y de las Comunicaciones en la Educación, SINTICE 2005* (editado por M. Ortega Cantero), páginas 259–266. Thomson, 2005b. ISBN 84-9732-437-4.

Gómez Martín, M. A., Gómez Martín, P. P., Palmier Campos, P. y González Calero, P. A. Not yet another visualization tool: Learning compilers for fun. En *8th International Symposium on Computers in Education (SIIE'06)* (editado por L. Panizo-Alonso, L. Sánchez-González, B. Fernández-Manjón y M. Llamas-Nistal), páginas 264–271. Universidad de León, León, Spain, 2006c. ISBN 84-9773-301-0.

Gómez Martín, P. P. y Gómez Martín, M. A. Fast application development to demonstrate computer graphics concepts. En *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '06)*, páginas 250–254. ACM Press, 2006. ISBN 1-59593-055-8.

Gómez Martín, P. P., Gómez Martín, M. A., Díaz Agudo, B. y González Calero, P. A. Opportunities for CBR in learning by doing. En *Proceedings of Case-Based Reasoning Research and Development, 6th International Conference on Case-Based Reasoning, ICCBR 2005* (editado por H. Muñoz Avila y F. Ricci), vol. 3620 de *Lecture Notes in Artificial Intelligence, subseries of LNCS*, páginas 267–281. Springer, Chicago, IL, US, 2005c. ISBN 978-3-540-28174-0. ISSN 0302-9743.

Gómez Martín, P. P., Gómez Martín, M. A. y González Calero, P. A. Javy: Virtual Environment for Case-Based Teaching of Java Virtual Machine. En *7th International Conference, Knowledge-Based Intelligent Information and Engineering Systems (KES 2003)*, vol. 2773 de *Lecture Notes in Artificial Intelligence, subseries of LNCS*, páginas 906–913. Springer, 2003. ISBN 3-540-40803-7. ISSN 0302-9743.

Gómez Martín, P. P., Gómez Martín, M. A. y González Calero, P. A. Learning-by-doing through metaphorical simulation. En *9th International Conference, Knowledge-Based Intelligent Information and Engineering Systems, KES 2005* (editado por R. Khosla, R. J. Howlett y L. C. Jain), vol. 3682 de *Lecture Notes in Artificial Intelligence, subseries of LNCS*, páginas 55–64. Springer, Melbourne, Australia, 2005d. ISBN 978-3-540-28895-4. ISSN 0302-9743.

Gómez Martín, P. P., Gómez Martín, M. A. y González Calero, P. A. Using metaphors in game-based education. En *Technologies for E-Learning and Digital Entertainment. Second International Conference of E-Learning and Games (Edutainment'07)* (editado por K. chuen Hui, Z. Pan, R. C. kit Chung, C. C. Wang, X. Jin, S. Göbel y E. C.-L. Li), vol. 4469, páginas 477–488. Springer, 2007b. ISBN 3-540-73010-9.

González Calero, P. A., Gómez Albarrán, M. y Díaz Agudo, B. A substitution-based adaptation model. En *Challenges for Case-Based*

Reasoning - Proceedings of the ICCBR'99 Workshops. University of Kaiserslautern, 1999.

González Calero, P. A., Gómez Martín, P. P. y Gómez Martín, M. A. Nuevas tecnologías, nuevas oportunidades. *Comunicación y Pedagogía*, vol. 216, páginas 71–76, 2006. ISSN 1136-7733.

Green, S. y Bavelier, D. Action-video-game experience alters the spatial resolution of vision. *Psychological Science*, vol. 18(1), páginas 88–94, 2007.

Gunderson, S., Jones, R. y Scanland, K. *The Jobs Revolution: Changing how America works*. The Grey Stone Group, 2004.

Hennessy, D. N. y Hinkle, D. Applying case-based reasoning to autoclave loading. *IEEE Expert*, vol. 7(5), páginas 21–26, 1992.

Hollnagel, E. The phenotype of erroneous actions: Implications for HCI design. En *Human Computer Interaction and Complex Systems* (editado por G. R. Weir y J. L. Alty). Academic Press, Inc., 1991. ISBN 0-12-742660-4.

Hollnagel, E. The phenotype of erroneous actions. *International Journal of Man-Machine Studies*, vol. 39(1), páginas 1–32, 1993. ISSN 0020-7373.

Holmevik, J. R. Compiling SIMULA: A historical study of technological generis. *IEEE Annals of the History of Computing*, vol. 16(4), páginas 25–37, 1994.

Howes, R. H. Undergraduate physics in the age of compassionate conservatism. Conferencia en la James Madison University, 2001.

Inoue, Y. Methodological issues in the evaluation of Intelligent Tutoring Systems. *Educational Technology Systems*, vol. 29(3), 2000.

Johnson, W. L., Rickel, J., Stiles, R. y Munro, A. Integrating pedagogical agents into virtual environments. *Presence: Teleoperators & Virtual Environments*, vol. 7(6), páginas 523–546, 1998.

de Jong, T. y van Joolingen, W. R. Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, vol. 68(2), páginas 179–201, 1998. ISSN 0034-6543.

Kafai, Y. B. The educational potential of electronic games: From games–to–teach to games–to–learn. En *Playing by the Rules: The cultural policy challenges of Video Games*. Chicago, EEUU, 2001.

Kafai, Y. B. Playing and making games for learning: Instructionist and constructionist perspectives for game studies. *Games and Culture*, vol. 1(1), páginas 34–40, 2006.

Kopec, D. y Thompson, R. B., editores. *Artificial Intelligence And Intelligent Tutoring Systems. Knowledge-based systems for teaching and learning*. Ellis Horwood, 1992. ISBN 0-13-048430-X.

Kumar, A. N. Model-based reasoning for domain modeling in a web-based intelligent tutoring system to help students learn to debug c++ programs. En *Proceedings of the 6th International Conference on Intelligent Tutoring Systems, ITS'02*, páginas 792–801. Springer-Verlag, 2002. ISBN 3-540-43750-9.

Labbo, L. D., Leu, D. J., Kinzer, C., Teale, W. H., Cammack, D., Kara-Soteriou, J. y Sanny, R. Teacher wisdom stories: Cautions and recommendations for using computer-related technologies for literacy instruction. *The Reading Teacher*, vol. 57(3), páginas 300–304, 2003. Disponible en `http://www.readingonline.org/electronic/elec_index.asp?HREF=/electronic/RT/11-03_column/index.html`.

Lacy, L. W. *OWL: Representing Information Using the Web Ontology Language*. Trafford Publishing, 2005. ISBN 1-4120-3448-5.

Lamarca Lapuente, M. J. *Hipertexto, el nuevo concepto de documento en la cultura de la imagen*. Tesis Doctoral, Facultad de Ciencias de la Información, Departamento de Biblioteconomía y Documentación, Universidad Complutense de Madrid, 2006. Disponible en `http://www.hipertexto.info/`.

Lee, J., Luchini, K., Michael, B., Norris, C. y Soloway, E. More than just fun and games: Assessing the value of educational video games in the classroom. En *CHI '04 extended abstracts on Human factors in computing systems*, páginas 1375–1378. ACM Press, 2004. ISBN 1-58113-703-6.

Leitão, J. M., Moreira, A., Santos, J. A., Sousa, A. A. y Ferreira, F. N. Evaluation of driving education methods in a driving simulator. En *Computer Graphics and Visualization Education GVE'99*. 1999.

Lepper, M. R. y Cordova, D. I. A desire to be taught: Instructional consequences of intrinsic motivation. *Motivation and Emotion*, vol. 16(3), páginas 187–208, 1992. ISSN 0146-7239.

Lester, J. C., Converse, S. A., Kahler, S. E., Barlow, S. T., Stone, B. A. y Bhogal, R. The persona effect: affective impact of animated pedagogical agents. En *Proceedings Human Factors in Computing Systems (CHI'97)*, páginas 359–366. 1997a.

Lester, J. C., Voerman, J. L., Towns, S. y Callaway, C. B. Cosmo: A life-like animated pedagogical agent with deictic believability. En *Working Notes of the IJCAI '97 Workshop on Animated Interface Agents: Making Them Intelligent*, páginas 61–69. 1997b.

Lester, J. C., Zettlemoyer, L. S., Grégoire, J. P. y Bares, W. H. Explanatory lifelike avatars: performing user-centered tasks in 3D learning environments. En *Proceedings of the 3rd annual conference on Autonomous Agents*, páginas 24–31. 1999.

Lewis Johnson, W., Shaw, E., Marshall, A. y LaBore, C. Evolution of User Interaction: The Case of Agent Adele. En *8th International Conference on Intelligent User Interfaces*, páginas 93–100. ACM Press, Miami, Florida, USA, 2003.

Lindholm, T. y Yellin, F. *The Java Virtual Machine Specification. 2nd Edition*. Addison-Wesley, 1999. ISBN 0-201-43294-3.

Long, J. Why not have fun while learning: Using programming games in software programming education. En *Proceedings of Information Systems Educator's Conference (ISECON 2006)*. 2006. ISSN 1542-7382.

Lowe, J. Computer-based education: Is it a panacea? *Journal of Research on Technology in Education*, vol. 34(2), páginas 163–171, 2001.

Maes, P. Agents that reduce work and information overload. *Communications of the ACM*, vol. 37(7), páginas 30–40, 1994. ISSN 0001-0782.

Malone, T. W. Toward a theory of intrinsically motivating instruction. *Cognitive Science: A Multidisciplinary Journal*, vol. 5(4), páginas 333–369, 1981a.

Malone, T. W. What makes computer games fun? *BYTE*, vol. 6, páginas 258–277, 1981b.

Malone, T. W. Making learning fun: A taxonomy of intrinsic motivations of learning. En *Conative and Affective Process Analyses* (editado por R. E. Snow y M. Farr), vol. 3 de *Aptitude, Learning, and Instruction*, capítulo 10, páginas 223–253. Lawrence Erlbaum, 1987. ISBN 0-89859721-8.

Mark, M. y Greer, J. Evaluation methodologies for Intelligent Tutoring Systems. *Journal of Artificial Intelligence and Education*, vol. 4(2-3), páginas 129–153, 1993.

Martin, B. y Mitrovic, A. Tailoring feedback by correcting student answers. En *Proceedings of the 5th International Conference on Intelligent Tutoring Systems (ITS 2000)* (editado por G. Gauthier, C. Frasson y K. VanLehn), vol. 1839 de *Lecture Notes in Computer Science*, páginas 383–392. Springer-Verlag, 2000. ISBN 3-540-67655-4.

Martínez Ortiz, I., Moreno Ger, P., Sierra, J. L. y Fernández Manjón, B. Production and deployment of educational videogames as

assessable learning objects. En *Proceedings of the First European Conference on Technology Enhanced Learning (ECTEL 2006)*, vol. 4227 de *Lectures Notes in Computer Science*, páginas 316–330. Springer, 2006.

Maslow, A. H. A theory of human motivation. *Psychological Review*, vol. 50, páginas 370–396, 1943. ISSN 0033-295X.

McFarlane, A., Sparrowhawk, A. y Heald, Y. Report on the educational use of games. TEEM: Teachers Evaluating Educational Multimedia, 2002.

McKenna, P. y Laycock, B. Constructivist or instructivist: pedagogical concepts practically applied to a computer learning environment. *SIGCSE Bulletin*, vol. 36(3), páginas 166–170, 2004. ISSN 0097-8418.

Mitchell, A. y Savill-Smith, C. *The use of computer and video games for learning: a review of the literature*. Learning and Skills Development Agency, 2004. ISBN 1-85338-904-8.

Molnar, A. Computers in education: A brief history. *THE Journal*, 1997.

Moreno, R., Mayer, R. E. y Lester, J. C. Life-like pedagogical agents in constructivist multimedia environments: Cognitive consequences of their interaction. En *Conference Proceedings of the World Conference on Educational Multimedia Hypermedia, and Telecommunications (ED-MEDIA)*, páginas 741–746. 2000.

Moreno Ger, P., Martínez Ortiz, I. y Fernández Manjón, B. The <e-game> project: Facilitating the development of educational adventure games. En *Cognition and Exploratory Learning in the Digital age (CELDA 2005)*, páginas 353–358. 2005.

Moundridou, M. y Virvou, M. Evaluating the persona effect of an interface agent in a tutoring system. *Journal of Computer Assisted Learning*, vol. 18(3), páginas 253–261, 2002.

Mulken, S. V., André, E. y Müller, J. The persona effect: How substantial is it? En *Proceedings of HCI on People and Computers XIII HCI'98*, páginas 53–66. Springer-Verlag, 1998. ISBN 3-540-76261-2.

Muller, J. P. *The Design of Intelligent Agents: A Layered Approach*. Springer-Verlag, 1997. ISBN 3-540-62003-6.

Nance, R. E. A tutorial view of simulation model development. En *Proceedings of the 15th conference on Winter simulation WSC'83*, páginas 325–332. IEEE Press, 1983.

Nass, C., Steuer, J. y Tauber, E. R. Computers are social actors. En *Proceedings of the SIGCHI conference on Human factors in computing systems CHI'94*, páginas 72–78. ACM Press, 1994. ISBN 0-89791-650-6.

Nelson, T. H. A file structure for the complex, the changing and the indeterminate. En *ACM 20th National Conference*, páginas 84–100. 1965.

Nogry, S., Jean-Daubias, S. y Duclosson, N. ITS evaluation in classroom: The case of Ambre-AWP. En *Intelligent Tutoring Systems*, páginas 511–520. 2004.

Nwana, H. S. Software agents: An overview. *Knowledge Engineering Review*, vol. 11(3), páginas 205–244, 1996.

Nygaard, K. y Dahl, O. J. The development of the SIMULA languages. En *The first ACM SIGPLAN conference on History of programming languages (HOPL-I)*, páginas 245–272. ACM Press, 1978.

Ong, J. y Ramachandran, S. Intelligent tutoring systems: The what and the how. Learning Circuits, publicado por la American Society of Training and Development. Disponible en `http://www.learningcircuits.org/2000/feb2000/ong.html`, 2000.

Paenza, A. *Matemática, ¿estás ahí?*. RBA, 2005. ISBN 84-7871-791-9.

Page, R. L. Brief history of flight simulations. En *Simulation Conference and Exhibition (SimTecT 2000)*, páginas 11–17. 2000.

Papert, S. *Constructionism Learning*, capítulo Introduction. Boston: MIT Laboratory, 1990.

Peinado Gil, F., Gómez Martín, P. P. y Gómez Martín, M. A. A game architecture for emergent story-puzzles in a persistent world. En *DIGRA 2005 - Changing Views: Worlds in Play - Electronic Proceedings*. 2005.

Perzov, A. y Kozminsky, E. The effect of computer games practice on the development of visual perception skills in kindergarten children. *Computer in the Schools*, vol. 6(3-4), páginas 113–122, 1989. ISSN 0738-0569.

Peña Marí, R. *De Euclides a Java: Historia de los algoritmos y de los lenguajes de programación*. NIVOLA libros y ediciones S.L., 2006. ISBN 84-96566-14-5.

Piaget, J. *The Construction of Reality in the Child*. New York: Basic Books, 1955.

Porter, B. Similarity assessment: Computation vs. representation. En *Proceedings of the CBR Workshop DARPA*. 1989.

Prendinger, H., Mayer, S., Mori, J. y Ishizuka, M. Persona effect revisited. En *Intelligent Virtual Agents (IVE'02)* (editado por T. Rist, R. Aylett, D. Ballin y J. Rickel), vol. 2792 de *Lecture Notes in Computer Science*, páginas 283–291. Springer, 2003. ISBN 3-540-20003-7.

Prensky, M. *Digital Game-Based Learning*. McGraw-Hill, 2001. ISBN 0-07-145400-4.

Pressey, S. L. A simple apparatus which gives tests and scores - and teaches. *School and Society*, vol. 23, páginas 373–376, 1926.

Pressey, S. L. A machine for automatic teaching of drill material. *School and Society*, vol. 25, páginas 549–552, 1927.

Pressey, S. L. A third and fourth contribution toward the coming "industrial revolution" in education. *School and Society*, vol. 36, páginas 668–672, 1932.

Provenzo, E. *Video Kids: Making Sense of Nintendo*. Harvard University Press, 1991. ISBN 0-674-93709-0.

Purushotma, R. Commentary: You're not studying, you're just.... *Language Learning & Technology*, vol. 9(1), páginas 80–96, 2005.

Pérez Martín, J. y Ruíz, J. I. Influencia del videojuego en la conducta y habilidades que desarrollan los videojugadores. *Edutec. Revista Electrónica de Tecnología Educativa*, (21), 2006. ISSN 1135-9250. Disponible en `http://edutec.rediris.es/Revelec2/revelec21/jperez.htm`. Existe una versión más detallada en `http://www.adese.es/pdf/Estudio_Videojuegos1.pdf`.

Quinn, C. N. Designing educational computer games. En *Proceedings of the IFIP TC3/WG3.2 Working Conference on the Seign, Implementation and Evaluation of Interactive Multimedia in University Settings*, páginas 45–57. Elsevier Science Inc., New York, NY, USA, 1994. ISBN 0-444-82077-9.

Recio, J. A., Díaz Agudo, B., Gómez Martín, M. A. y Wiratunga, N. Extending jCOLIBRI for textual CBR. En *Proceedings of Case-Based Reasoning Research and Development, 6th International Conference on Case-Based Reasoning, ICCBR 2005* (editado por H. Muñoz-Avila y F. Ricci), vol. 3620 de *Lecture Notes in Artificial Intelligence, subseries of LNCS*, páginas 421–435. Springer, Chicago, IL, US, 2005. ISBN 978-3-540-28174-0. ISSN 0302-9743.

Repenning, A. y Lewis, C. Playing a game: The ecology of designing, building and testing games as educational activities. En *World Conference on Educational Multimedia, Hypermedia & Telecommunications: ED-Media 2005*. 2005.

Reyes, R. L. y Sison, R. C. Representing and indexing cases in CBR-Tutor. En *Ninth International Conference on Computers in Education*, páginas 242–249. 2001.

Reyes, R. L. y Sison, R. C. Case retrieval in CBR-Tutor. En *International Conference on Computers in Education (ICCE'02)*, vol. 2, páginas 879–883. IEEE Computer Society, 2002. ISBN 0-7695-1509-6. Del 3 al 6 de diciembre.

Rice, J. W. Evaluating the suitability of video games for k-12 instruction. Artículo presentado en Exploring the Vision, Association for Educational Communications and Technology 2005 International Convention, Orlando, FL, 2005a.

Rice, J. W. Video games in the classroom? What the research is telling us. *TechEdge: journal of the Texas Computer Education Association*, 2005b.

Rich, E. User modelling via stereotypes. *Cognitive Science*, vol. 3, páginas 319–354, 1979.

Rich, E. Users are individuals: Individualizing user models. *International Journal of Man-Machine Studies*, vol. 18(3), páginas 199–214, 1983.

Rickel, J. y Johnson, W. L. Animated agents for procedural training in virtual reality: perception, cognition and motor control. *Applied Artificial Intelligence*, vol. 13(4), páginas 343–382, 1999.

Rickel, J. W. Intelligent computer-aided instruction: A survey organized around system components. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19(1), páginas 40–57, 1989.

Rieber, L. P. Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational Technology Research & Development*, vol. 44(2), páginas 43–58, 1996. ISSN 1042-1629.

Riesbeck, C. K. y Schank, R. C. *Inside Case-Based Reasoning*. Lawrence Erlbaum, 1989. ISBN 0-89859-767-6.

Rist, R. y Hewer, S. *Implementing Learning Technology*, capítulo What is learning technology? Some definitions. Learning Technology Dissemination Initiative, 1996. ISBN 0-9528731-0-9.

Rodríguez Andrés, A. Los determinantes socioeconómicos del delito en España. *Revista Española de Investigación Criminológica*, vol. 1, 2003. ISSN 1696-9219.

Rollings, A. y Morris, D. *Game Architecture and Design*. New Riders, Indianapolis, 2003. ISBN 0-7357-1363-4.

Rutherford, A. B. F. Skinner's technology of behavior in american life: From consumer culture to counterculture. *Journal of the History of the Behavioral Sciences*, vol. 39(1), páginas 1–23, 2003.

Santituste Bermejo, V. *Formación de profesores de educación secundaria*, capítulo El aprendizaje cognitivo, páginas 245–256. Cyan, Proyectos y Producciones Editoriales, S.A., 2003a. ISBN 84-8198-453-1.

Santituste Bermejo, V. *Formación de profesores de educación secundaria*, capítulo El condicionamiento operante y sus aportaciones a la educación, páginas 229–240. Cyan, Proyectos y Producciones Editoriales, S.A., 2003b. ISBN 84-8198-453-1.

Schery, T. y O'Connor, L. Language intervention: Computer training for young children with special needs. *British Journal of Educational Technology*, vol. 28(4), páginas 271–279, 1997.

Seifert, R. V. *Microcomputers in Adult Education*. Routledge Kegan & Paul, 1986. ISBN 0-7099-3944-2.

Sharman, R., Kishore, R. y Ramesh, R., editores. *Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems*. Springer–Verlag, 2007. ISBN 978-0387-37019-4.

Shaw, E., Lewis Johnson, W. y Ganeshan, R. Pedagogical agents on the web. En *Third annual conference on Autonomous Agents*, páginas 283–290. ACM Press, 1999.

Siang, A. C. y Rao, R. K. Theories of learning: A computer game perspective. En *Proceedings of the IEEE Fifth International Symposium on Multimedia Software Engineering (ISMSE'03)*, páginas 239–245. 2003. ISBN 0-7695-2031-6.

Siemer, J. y Angelides, M. C. Evaluating intelligent tutoring with gaming-simulations. En *Proceedings of the 27th conference on Winter simulation (WSC '95)* (editado por C. Alexopoulos, K. Kang, W. R. Lilegdon y D. Goldsman), páginas 1376–1383. ACM Press, New York, NY, USA, 1995. ISBN 0-7803-3018-8.

Silvern, S. B. y Williamson, P. A. The effects of game play on young children's aggression, fantasy, and prosocial behavior. *Journal of Applied Social Psychology*, vol. 8(4), páginas 456–462, 1987.

Skinner, B. F. *The Behavior of organisms*. Appleton-Century, 1938. Traducción al español: *La conducta de los organismos: un análisis experimental*, Barcelona, Fontanella, 1979.

Skinner, B. F. Teaching machines. *Science*, vol. 128, páginas 969–977, 1958.

Skinner, B. F. *The Technology of Teaching*. Appleton Century Crofts, 1968. Traducción al español: *Tecnología de enseñanza*, Labor, 1982.

Skinner, B. F. *Notebooks*. Prentice-Hall, 1980. ISBN 0-13-624106-9.

Skinner, B. F. *A Matter of Consquences*. Knopf, 1983.

Sleeman, D. H. y Brown, J. S., editores. *Intelligent Tutoring Systems*. Academic Press, London, 1982.

Sloane, H., Gordon, H. y Gunn, C. *Evaluating Educational Software: A Guide for Teachers*. Prentice Hall, 1989. ISBN 0-13-298571-3.

Smith, L. M. B. F. Skinner. *Perspectivas: Revista Trimestral de Educación Comparada*, vol. 24(3,4), páginas 529–542, 1994. Reeditado en 1997 como capítulo de *Thinkers on Education*, UNESCO Publishing, ISBN 92-3-103398-0.

del Soldato, T. y du Boulay, B. Implementation of motivational tactics in tutoring systems. *Journal of Artificial Intelligence in Education*, vol. 6(4), páginas 337–378, 1995. ISSN 1043-1020.

Squire, K. Video games in education. *International Journal of Intelligent Simulations and Gaming*, vol. 2(1), páginas 49–62, 2003.

Squire, K. *Replaying history: learning world history through playing Civilization III*. Tesis Doctoral, Faculty of the School of Education for the degree Doctor of Philosophy in the Instructional Systems Technology Department Indiana University, 2004.

Squire, K. y Barab, S. Replaying history: engaging urban underserved students in learning world history through computer simulation games. En *Proceedings of the 6th international conference on Learning sciences (ICLS '04)*, páginas 505–512. International Society of the Learning Sciences, 2004.

Squire, K. y Jenkins, H. Harnessing the power of games in education. *Insight*, vol. 3(1), páginas 5–33, 2003.

Stone, B. y Lester, J. C. Dynamically sequencing an animated pedagogical agent. En *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, páginas 424–431. 1996.

Stottler, D., Harmon, N. y Michalak, P. Transitioning an ITS developed for schoolhouse use to the fleet: TAO ITS, a case study. En *Proceedings of the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC 2001)*. 2001a.

Stottler, R. H. Tactical action officer intelligent tutoring system (TAO ITS). En *Proceedings of the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC 2000)*. 2000.

Stottler, R. H., Fu, D., Ramachandran, S. y Jackson, T. Applying a generic Intelligent Tutoring System (ITS) authoring tool to specific military domains. En *Proceedings of the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC 2001)*. 2001b.

Subrahmanyam, K. y Greenfield, P. M. Computer games for girls: what makes them play? En *From Barbie to Mortal Kombat: Gender and Computer Games* (editado por J. Cassell y H. Jenkins), páginas 453–462. The MIT Press, 1998. ISBN 0-262-03258-9.

Taylor, J. L. y Walford, R. *Learning and the Simulation Game*. Milton Keynes: Open University Press, 1978. ISBN 0-335-00237-4.

Taylor, R., editor. *The Computer in the School: Tutor, Tool, Tutee*. Teachers College Press, 1980. ISBN 0-8077-2611-7.

The Economist. Defending video games: Breeding evil? *The Economist (print version) 4 de Agosto de 2005*, Disponible en `http://www.economist.com/PrinterFriendly.cfm?story_id=4247084`.

Thomas, S., Schott, G. y Kambouri, M. Designing for learning or designing for fun? setting usability guidelines for mobile educational games. En *Proceedings of MLEARN 2003: Learning with Mobile Devices*. Londres, 2003.

Thorndike, E. L. *Education: A first book*. The MacMillan Company, 1912.

Trindade, J. F., Fiolhais, C., Gil, V. y Teixeira, J. C. Virtual environment of water molecules for learning and teaching science. En *Proceedings of the Computer Graphics and Visualization Education'99 – (GVE'99)*, páginas 153–158. 1999.

UNESCO y UNICEF. *Children out of School: Measuring Exclusion from Primary Education*. UNESCO - Institute for Statistics, 2005. ISBN 92-9189-026-X.

Vargas, E. A. y Vargas, J. S. Programmed instruction: What it is and how to do it. *Journal of Behavioral Education*, vol. 1(2), páginas 235–251, 1991.

Vince, J. *Virtual Reality Systems*. Addison Wesley Longman, 1995. ISBN 0-201-87687-6.

Vázquez Gómez, G. *Formación de profesores de educación secundaria*, capítulo Concepto y características de la educación, páginas 21–28. Cyan, Proyectos y Producciones Editoriales, S.A., 2003. ISBN 84-8198-453-1.

Watson, J. B. Psychology as a behaviorist views it. *Psychological Review*, vol. 20, páginas 158–177, 1913. ISSN 0033-295X.

Weber, G. y Brusilovsky, P. ELM-ART: An adaptive versatile system for Web-based instruction. *International Journal of Artificial Intelligence in Education*, vol. 12(4), páginas 351–384, 2001.

Wenger, E. *Artificial intelligence and tutoring systems: computational and cognitive approaches to the communication of knowledge*. Morgan Kaufmann Publishers Inc., 1987. ISBN 0-934613-26-5.

Wikipedia (Producto_interior_bruto-es). Entrada: "Producto interior bruto". Disponible en `http://es.wikipedia.org/wiki/Producto_interno_bruto` (último acceso, Octubre, 2007).

Wolf, M. J. P. *The Medium of the Video Game*. University of Texas Press, 2002. ISBN 0-292-79150-X.

Woolf, B. *Artificial Intelligence And Intelligent Tutoring Systems. Knowledge-based systems for teaching and learning*, capítulo Hypermedia in Education and Training, páginas 97–109. Ellis Horwood, 1992a. ISBN 0-13-048430-X.

Woolf, B. *Encyclopedia of Artificial Intelligence*, capítulo AI in Education, páginas 434–444. John Wiley & Sons, Inc., 1992b. ISBN 0-471-50305-3.

Xing, W. The more they play, the more they lose. *China Daily, 10 de Abril*, 2007. Disponible en `http://www.chinadaily.com.cn/cndy/2007-04/10/content_846614.htm` (último acceso, Octubre, 2007).

Lista de acrónimos

ADESE ..... Asociación Española de Distribuidores y Editores de Software de Entretenimiento

APX ..... *Atari Program Exchange*, Intercambio de Programas de Atari

CAD ..... *Computer Aided Design*, Diseño asistido por ordenador

CBR ..... *Case-based reasoning*, Razonamiento basado en casos

DAML ..... *DARPA Agent Markup Language*, Lenguaje de Marcas de Agentes de DARPA

DARPA ..... *Defense Advanced Research Projects Agency*, Agencia de Proyectos de Investigación Avanzados de Defensa

DTD ..... *Document Type Definition*, Definición de tipo de documento

EPA ..... Encuesta de Población Activa

ESA ..... *Entertainment Software Association*, Asociación de software de entretenimiento

FCA ..... *Formal Concept Analysis*, Análisis Formal de Conceptos

FPS ..... *First Person Shooters*, Acción en primera persona

GrimE ..... *Grim Edit*

INCE ..... Instituto Nacional de Calidad y Evaluación

INE ..... Instituto Nacional de Estadística

INECSE ..... Instituto Nacional de Evaluación y Calidad del Sistema Educativo

INEM ..... Instituto de Empleo

IR ..... *Information Retrieval*, recuperación de información

JIT ..... *Just In Time*, en el momento

JVM ..... *Java Virtual Machine*, Máquina virtual de Java

KI-CBR ..... *Knowledge Intensive Case-Based Reasoning*, Razonamiento basado en casos rico en conocimiento

LMS ..... *Learning Management System*, Sistema Gestor de Aprendizaje

LOCE..... Ley Orgánica de Calidad de la Educación

MEC..... Ministerio de Educación y Ciencia

OIL ..... *Ontology Inference Layer*, Capa de Inferencia con Ontologías

OLPC..... *One Laptop per Child*, Un portátil para cada niño

ONU ..... Organización de las Naciones Unidas

OWL..... *Web Ontology Language*, Lenguaje de Ontologías para la Web

PARC..... *Palo Alto Research Center*, Centro de Investigación de Palo Alto

PDA ..... *Personal Digital Assistant*, Ayudante Digital Personal

PEGI ..... *Pan European Game Information*, Información paneuropea sobre juegos

PIB ..... Producto Interior Bruto

RAE ..... Real Academia Española

SCUMM ..... *Script Creation Utility for Maniac Mansion*, Utilidad de Creación de Guiones para Maniac Mansion

SMS ..... *Short Message Service*, Servicio de Mensajes Cortos

TIC ..... Tecnologías de la Información y la Comunicación

UNESCO ..... *United Nations Educational, Scientific and Cultural Organization*, Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura

W3C..... *World Wide Web Consortium*, Consorcio de la WWW .

WIMP ..... *Windows, Icon, Mouse & Pointer*

El general le diría algo ingenioso,
pero ahora se lo impide la emoción.

Ciudadano X (sobre el minuto 90)
Chris Gerolmo (1995)

–Y adiós, que ya viene el alba.
Y dando a sus mulas, no
atendió a más preguntas.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes