Generación automática de contenido para un nuevo juego basado en el problema de los tres cuerpos

Alejandro Gutierrez Alcoba[1], Raúl Lara-Cabrera[2] and Antonio J. Fernández-Leiva[2]

[1] Departamento de Arquitectura de Computadores, Universidad de Málaga
agutierrez@ac.uma.es
[2] Departmento de Lenguajes y Ciencias de la Computación, Universidad de Málaga
{raul,afdez}@lcc.uma.es

Resumen Este trabajo presenta un algoritmo de generación de contenido por procedimientos capaz de crear mapas completos para un videojuego que simula fenómenos físicos. El algoritmo evolutivo desarrollado intenta mejorar la dificultad de los mapas. Además, gracias al uso de poblaciones estructuradas, el algoritmo puede construir mapas que supongan un desafío tanto a los jugadores más avanzados como a los más inexpertos.

1. Introducción

Con el paso del tiempo los videojuegos se han ido convirtiendo en uno de los sectores más importantes y lucrativos de la industria del entretenimiento. Solo en 2011 en los Estados Unidos el sector experimentó unos ingresos de más de 16 billones de dólares [1]. Por otra parte, los costes económicos para producir un videojuego son muy elevados: el desarrollo es un proceso lento y precisa de un amplio equipo de profesionales heterogéneo y muy cualificado. Por tanto, cualquier mejora que consiga optimizar el tiempo de desarrollo o los recursos disponibles es siempre bienvenida.

El área de la inteligencia computacional sirve de aplicación directa al sector del videojuego [10], [11]. La generación de contenido por procedimientos, o PCG en inglés, Procedural Content Generation, consiste en la generación automática, mediante algoritmos en lugar de forma manual, de contenido para videojuegos. Este contenido generado algorítmicamente puede ser muy diverso, pudiendo referirse a los terrenos del juego, sus mapas, niveles completos, características de armas u otros objetos, incluso ciertas reglas del propio juego. Puede tratarse desde contenido totalmente necesario para avanzar en el desarrollo del videojuego hasta aquel que es puramente decorativo. Eso sí, los algoritmos utilizados deben garantizar la generación de contenido que cumpla ciertos requisitos de calidad, adaptándose al tipo de contenido que se pretendiese crear manualmente. Por ejemplo, en el caso de tratarse de contenido necesario, un buen fin en mente

del equipo desarrollador de un videojuego sería adaptar el contenido a distintos estilos de juego.

Son muchas las ventajas de producir contenido de videojuegos mediante algoritmos PCG. En primer lugar puede permitirnos reducir sustancialmente el consumo de memoria del juego, que si bien en la actualidad puede tener un caracter secundario motivó en el pasado el uso de estas técnicas, pudiendo verse en este caso como una forma de comprimir el contenido del juego hasta el momento que sea necesario. Otro motivo importante para el uso de estos algoritmos es, como ya se ha mencionado, el elevado coste que tiene en muchos casos la generación de cierto contenido de los juegos de forma manual.

Además, si como se ha dicho antes se asegura que el contenido generado por PCG cumple con ciertos criterios, como ajustarse a la habilidad de un jugador, el nuevo contenido puede suponerle un reto constante. Si esta generación de contenido adaptado resulta siempre diversa y se genera de forma online, es decir, al mismo tiempo que el juego se ejecuta, pueden conseguirse auténticos juegos infinitos, presentando constantemente a cada tipo de jugador nuevos y distintos retos que ha de superar.

2. Antecedentes

Cuando se trata de generar contenido automático para videojuegos pueden hacerse muchas distinciones globales en lo que respecta a los procedimientos a seguir. Siguiendo la taxonomía propuesta en [18], en primer lugar la generación de contenido puede ser tanto online, durante la ejecución del juego (con las ventajas que esto puede tener, ya comentadas previamente) como offine, durante la fase de desarrollo del juego.

El contenido generado puede ser considerado como contenido necesario para poder progresar en el juego, en cuyo caso se ha de asegurar que el contenido es correcto (en el sentido de no proponer objetivos imposibles al jugador), o bien opcional, como pueden ser los elementos decorativos y que por lo general pueden ser mucho menos restrictivos.

Un algoritmo PCG puede generar todo el contenido a partir de semillas aleatorias. En el otro extremo, el contenido se genera a partir de un vector de parámetros. Otra pregunta que cabe hacerse es en qué grado el algoritmo debe ser determinista o por el contrario estocástico.

Según los objetivos que se pretendan satisfacer, la generación del contenido puede hacerse de forma constructiva, asegurando que el contenido es válido en primera instancia, o bien mediante generación y test. En este caso el contenido generado pasa por un proceso de validación y en caso de no superar el test, todo o parte del contenido generado es descartado y vuelto a generar.

Un tipo particular de algoritmos de generación de contenido automático en auge actualmente es el basado en la búsqueda en el espacio de posibles soluciones del contenido a generar, Search-based procedural content generation (SBPCG). Es un caso particular de generación de contenido en el que se lleva a cabo un proceso de generación y test del contenido, pero no sencillamente aceptando

o rechazando el contenido, sino asignándole un valor real (o varios, en el caso multiobjetivo) que mida el grado de bondad del contenido creado.

En este tipo concreto de PCG es muy común el uso de algoritmos evolutivos [3] o genéticos [12], aunque no necesariamente es este el proceso a seguir y otros tipos de técnicas pueden ser utilizadas con el mismo planeamiento SBPCG.

Son muchos los ejemplos de artículos relacionados con esta materia. M. Hendrikx et al. [8] hace un profundo análisis sobre la diversidad del contenido que puede generarse por estos medios. La investigación en este campo es muy activa en la actualidad y pueden encontrarse multitud de artículos relacionados, entre los que podemos citar algunos: Un ejemplo clásico es el de la generación de niveles para juegos de plataformas. Noor Shaker et al., proponen en [14] un sistema para adaptar al estilo de juego de distintos jugadores los parámetros para el diseño de niveles automáticos para un clon de Super Mario Bros. Con respecto a este juego, Pedersen et al. [13] investigan la relación entre los parámetros del diseño de niveles con el estilo de juego y la experiencia de juego causada (diversión, frustración, ...).

Otro contenido que se puede evolucionar son los mapas. Frade et al. [5], proponen una función de fitness para generar terrenos accesibles. En [4] evolucionan mapas que han sido usados en el videojuego chapas. Por su parte, Julian Togelius et al., diseñaron un algoritmo evolutivo multiojbetivo SBPCG para generar mapas para juegos de estrategia [17] y [16]. El mismo autor y otros muestran en [15] un algoritmo que sirve para generar circuitos de un juego de carreras a partir de un vector de parámetros. En [9], por Hom y Marks, se generan reglas de un juego de mesa para dos personas, persiguiendo además la propiedad de equilibrio entre ambos jugadores.

También existen numerosos ejemplos para contenido opcional del juego, como pueden ser las armas que el jugador puede llevar consigo. Hastings et al. [6], [7] diseñaron un algoritmo SBPCG para el juego Galactic Arms Race. El fitness de las armas creadas se calcula teniendo en cuenta el tiempo que los jugadores la usaban, pudiendo así medir la satisfacción del jugador con el contenido de manera natural y sin requerir la atención explícita de los jugadores en la evaluación.

3. Descripción del problema

Uno de los principales objetivos en mente cuando se pretende generar contenido con técnicas SBPCG (obviando la motivación inherente de no necesitar la labor humana para producirlo) consiste en tratar de maximizar la diversión que produce en los jugadores. No obstante, con el conocimiento actual sobre este aspecto, las medidas que se suelen proponer suelen ser subjetivas o se dejan caer en presunciones no contrastadas.

Antes de proseguir con los detalles del algoritmo PCG diseñado es necesario explicar el funcionamiento del juego. Si bien este debe considerarse como una herramienta para el resto de este trabajo, es necesario introducirlo con detalle, sobre todo para comprender como se definen las funciones de aptitud adaptadas al juego.

3.1. El juego

Dentro del campo de la física y las matemáticas existe un problema clásico llamado *problema de los tres cuerpos*, que puede generalizarse a n cuerpos: este último consiste en determinar, en cualquier instante, las posiciones y velocidades de n partículas regidas por la ley de gravitación universal de Newton, partiendo de cualesquiera condiciones iniciales de posición y velocidad. Dicho problema no tiene solución analítica (para $n \geq 3$) y en general solo puede resolverse con métodos de integración numérica [2]. El juego se inspira en una simulación de este conocido problema, mas con ciertas particularidades que permiten convertir una simulación interactiva de este problema en un entorno jugable. En primer lugar, la simulación se hace en dos dimensiones, ya que hacerlo en un espacio tridimensional solo lo complicaría de forma innecesaria. Lo segundo, y más importante, es que en la simulación tan solo una de todas las partículas recibirá la fuerza gravitatoria del resto. Esta partícula será la única con la que el jugador podrá interactuar, pudiendo cambiar su vector de velocidad en determinados momentos, para guiarla por la pantalla. El resto de partículas permanecerán siempre en reposo en la misma posición.

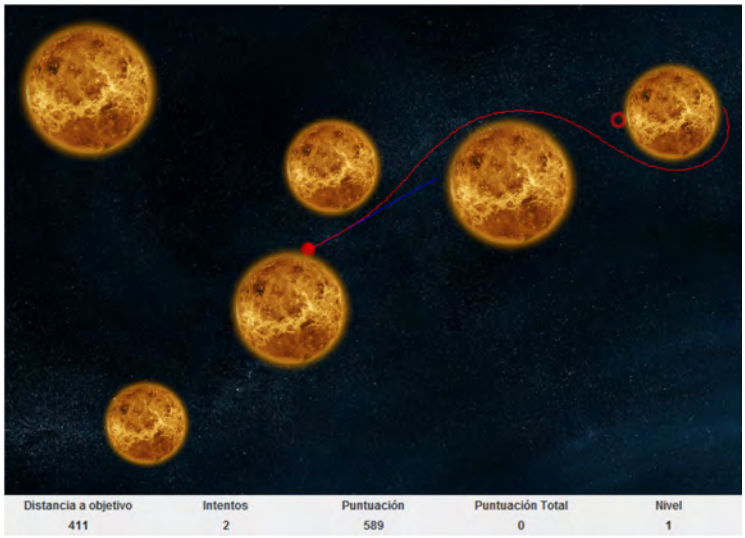

Figura 1. Ejemplo de un nivel

En la figura 1 se muestra un ejemplo del juego en acción. Como puede observarse, sobre la pantalla aparecen situadas en distintas posiciones n partículas, cada una de ellas con una masa asociada, que se representarán en pantalla como planetas con un radio determinado por la masa de cada una. Al comenzar el juego, sobre uno de los planetas del nivel, se encontrará otra partícula, de dimensiones mucho más reducidas que el resto, en estado de reposo. Esta partícula

es aquella con la que el usuario interactuará y recibirá la fuerza gravitatoria de todos los planetas del nivel.

Se propone un objetivo simple para el jugador: hacer que la partícula quede en estado de reposo lo más cerca posible a una determinada posición del plano, marcada con un círculo (como puede verse en la figura 1), sobre otro de los planetas del nivel distinto a aquel donde se posicionaba inicialmente la partícula, realizando lanzamientos aplicando un vector de fuerza a la bola roja. Por cada nivel el jugador dispone de tres intentos para acercarse lo máximo posible a la posición objetivo. La puntuación que obtenga será mayor cuanto más cerca quede parada la partícula y del número de intentos que haya requerido para dejar en reposo a la partícula sobre el planeta objetivo.

4. Generador automático de mapas

El generador de mapas basa su funcionamiento en un algoritmo evolutivo de estado estacionario y población estructurada en tres clases de igual tamaño. El primero contiene aquellos individuos mejor adaptados a la función objetivo utilizada. Como ambas están encaminadas a mejorar la dificultad en esta primera se encuentran los individuos más difíciles, en la segunda están los medios y en la última los más sencillos. De esta forma con una sola ejecución del algoritmo se consigue un amplio espectro de mapas del juego adaptados a todos los niveles.

4.1. Operadores de mutación y cruce

Con respecto a la mutación de los individuos, en un nivel con n planetas, el número de genes correspondientes a posición X, posición Y o masa asciende a $3n$. Por tanto, en la fase de mutación se ha establecido que cada uno de esos genes pueda ser modificado con una probabilidad $p = \frac{1}{3n}$. La elección de este valor no es caprichosa: al hacerlo de esta forma el número de genes de este tipo que son mutados en cada nivel sigue una distribución binomial $X \sim B(3n, \frac{1}{3n})$. Así, la cantidad media de genes los genes correspondientes a los planetas que son mutados resulta ser

$$E(X) = 3n\frac{1}{3n} = 1$$

En el caso de que uno de los genes de posición sea escogido la variación que se le hace es sumarle un valor x que siga una distribución normal de media 0 y varianza 50. Con la masa se ha seguido la misma idea de usar una distribución normal, de forma que el planeta en cuestión pueda encoger o agrandar su masa (y radio) una cierta cantidad. Y aunque por supuesto no son comparables las magnitudes de posición y masa, la varianza escogida para la normal ha sido la misma que en el caso anterior, 50. En cuanto a los cuatro genes correspondientes a la posición en el nivel de la partícula y el objetivo, se ha optado por mutarlos cada uno con probabilidad 0,15.

El método de la mutación, tras haber pasado uno a uno por cada gen del nivel y cambiando aquellos seleccionados, comprueba si en el proceso se ha producido un efecto no deseado en el nivel, como hacer que un planeta sobresalga de los límites establecidos en la pantalla o que dos planetas queden intersecados o muy cerca. Al igual que se hace en el cruce, si algo así sucede, se vuelve a dejar el nivel como se encontraba inicialmente y se repite el proceso hasta que la mutación se realice correctamente.

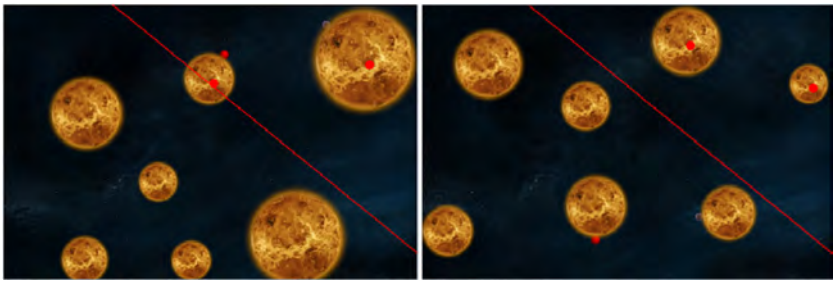

Figura 2. Ejemplo de dos niveles seleccionados

El operador de cruce de dos individuos que se propone resulta original, ya que se inspira en la recombinación en un punto pero de forma geométrica. Lo primero que hace es calcular una recta aleatoria pero que se asegure de cortar el área del nivel en dos partes. Eligiendo cuatro puntos al azar se calcula la ecuación de la recta y, posteriormente, se guardan por separado los planetas que han quedado en la parte superior de los que han quedado en la inferior. El punto que determina si un planeta queda en la parte superior o inferior es el centro de dicho planeta, es decir, el valor de su posición. Si en cada una de las partes ha quedado al menos un planeta y coinciden el número de planetas que hay en la parte superior y en la inferior de ambos individuos seleccionados, se toma el corte como válido (ver figura 2).

El hacer el corte de esta manera generaliza el comportamiento que tendría la clásica recombinación en un punto con la que, si por ejemplo consideramos los planetas ordenados por el orden lexicográfico (uno de los pocos que tendría sentido para el propósito que nos ocupa) los cortes serían, geométricamente, como líneas prácticamente horizontales o verticales según se considere ordenar primero por la coordenada Y o por la X. Aún queda por definir como se generan los dos individuos descendientes a partir de los primeros. Es fácil: la parte superior del primer padre con la parte inferior del segundo conforman el primer hijo. Análogamente se genera el segundo. En las figuras 2 y 3 se muestra un ejemplo de cruce. Se ha dibujado una línea roja con la recta que sirvió para separar los planetas en ese caso. Para diferenciarlos más fácilmente, los planetas de la parte superior aparecen con un círculo rojo pintado en su centro. Como se puede apreciar, en el primer hijo dos planetas casi se intersecan al combinar

parte de un padre y la del otro. En un caso como este se repetiría nuevamente el cruce para intentar dar una descendencia correcta. La condición impuesta de la separación mínima entre los planetas es en ocasiones estricta, por lo que si tras un número de intentos el cruce no se produce correctamente se aborta el proceso.

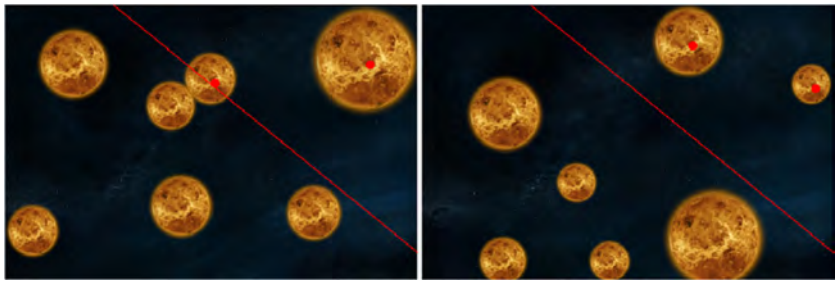

Figura 3. Ejemplo de descendencia de la figura 2

4.2. Función de aptitud

Para este trabajo se han propuesto dos funciones de aptitud distintas que representan propiedades del nivel que en un caso al maximizar y en otro al minimizar, hacen más difícil un nivel. Dentro de todo lo expuesto en este trabajo, estas son las medidas más concretas y particulares al juego tratado y que por tanto son menos extrapolables a otro tipo de juegos.

Método de las intersecciones La primera idea es bien simple: consiste en poner "tierra de por medio" entre la partícula y el planeta en que se encuentra el objetivo. El método calculará, en primer lugar, la ecuación de la recta que pasa por el punto central de la partícula y del planeta objetivo. Posteriormente comprueba, entre los planetas intermedios entre aquel donde esté la partícula y donde esté el objetivo (así se evita comprobar algunos), si cada uno de ellos interseca con la recta y cual es esa distancia. Si la recta definida por los puntos mencionados tiene por ecuación $y = mx + n$ y la de la circunferencia del planeta tratado en el momento es $(x-a)^2 + (y-b)^2 = R^2$, los puntos de corte, en caso de existir, son:

$$\begin{cases} x_1 = \dfrac{\sqrt{R^2m^2 + R^2 - a^2m^2 + 2abm - 2amn - b^2 + 2bn - n^2}}{m^2+1} + \dfrac{a + bm - mn}{m^2+1} \\ y_1 = \dfrac{a + bm - mn}{m^2+1} \end{cases}$$

y con el otro signo:

$$\begin{cases} x_1 = -\dfrac{\sqrt{R^2m^2 + R^2 - a^2m^2 + 2abm - 2amn - b^2 + 2bn - n^2}}{m^2+1} + \dfrac{a + bm - mn}{m^2+1} \\ y_1 = \dfrac{a + bm - mn}{m^2+1} \end{cases}$$

El valor de *fitness* de este método será la suma de todas estas intersecciones. Resulta interesante que el uso del algoritmo genético con esta función de aptitud aumenta también de forma indirecta la distancia entre la posición inicial de la partícula y del planeta objetivo.

Como observación, si sobre el nivel no se impusieran condiciones como respetar una mínima distancia entre los planetas y solo se tuviera en cuenta que no se intersecaran parece claro cual sería la forma típica que tendría un nivel con un fitness máximo global (que además se alcanzaría con mucha más facilidad): los planetas (no tendrían por qué ser todos) estarían alineados con una de las diagonales de la pantalla, sin un solo hueco entre ellos y la partícula y el objetivo se encontrarían en esquinas de la pantalla opuestas.

Afortunadamente las restricciones impuestas a los niveles impide no solo que algo así suceda sino que permiten mucha más variedad en los niveles generados. Aún así existe cierta tendencia a ver varios planetas en torno a una diagonal , pero podría solucionarse por completo, con un algoritmo multiobjetivo que premiase la poca correlación lineal entre los puntos centrales de los planetas.

Minimizar la atracción en el planeta objetivo Otra forma de a nadir dificultad al objetivo de situar la partícula sobre el planeta objetivo es minimizar su fuerza de atracción. Una forma simple de hacerlo sería minimizar la masa del planeta objetivo al mismo tiempo que se aumenta la del resto. Por ejemplo, podría definirse como función objetivo la masa relativa del planeta objetivo, m con respecto a la suma de los demás, M

$$\text{fitness} = \frac{m}{M}$$

No obstante y aunque naturalmente funciona, es evidente que no es necesario utilizar un algoritmo evolutivo para generar niveles automáticamente que satisfagan la minimización de esta medida. Se podría dise nar como algoritmo PCG un método que construyese directamente un nivel con estas características. Y además, esta técnica daría lugar a niveles prácticamente iguales en cuanto a las masas de sus planetas y aún así no se tendrían en cuenta las posiciones de estos para poder controlar la dificultad de los niveles con mayor precisión.

Por ello se ha pensado otra forma de medir la atracción del planeta objetivo con una función que tuviera en cuenta las masas de todos los planetas pero también sus posiciones. En la figura 6 se muestra un ejemplo del proceso que se realiza en el método. Para hacer la medición se posiciona en primer lugar una partícula auxiliar a cierta distancia d del planeta objetivo (superior al radio de este). Para ese módulo d se consideran raíces n-ésimas (por defecto se toma

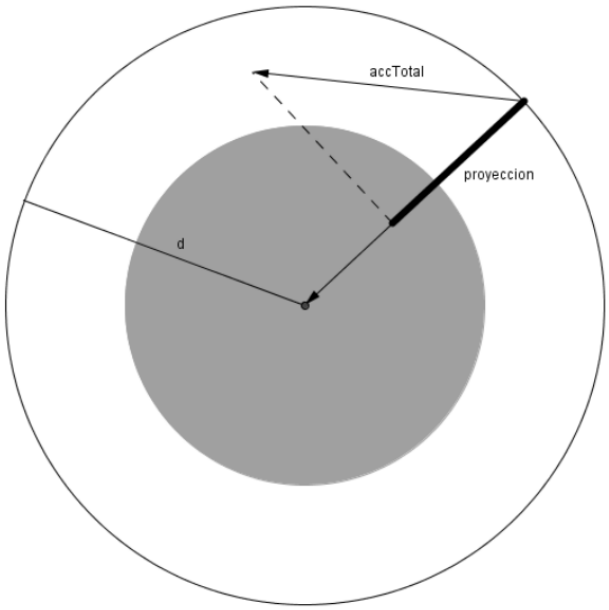

Figura 4. Proyección de la aceleración

$n = 50$) y en cada una de estas posiciones alrededor del planeta a distancia d se calcula el vector de aceleración (etiquetado en la figura 4 como *accTotal*) para finalmente calcular su proyección sobre el vector definido entre el punto considerado en ese momento y el centro del planeta objetivo. El valor de la proyección indica con su signo si en ese punto la tendencia es de atracción o repulsión y con qué magnitud.

Evaluación basada en la simulación Se ha dise nado también otra función de aptitud basada en la simulación directa sobre el juego. El método simula el lanzamiento de la partícula con multitud de vectores de velocidad diferentes, en módulo y ángulo, eliminando por supuesto la limitación del juego de iteraciones fijas por segundo y su parte gráfica. Al final de cada lanzamiento comprueba si la partícula ha quedado en estado de equilibrio sobre el planeta objetivo. En caso afirmativo suma 1 a una variable local (para calcular al final la frecuencia relativa de aciertos) y hace una estimación de hasta que punto la forma de llegar al planeta objetivo fue fortuita (a mayor distancia recorrida por la partícula y mayor número de colisiones con otros planetas durante el trayecto se considera el acierto más debido al azar que a la intuición humana). En concreto, la fórmula usada es la siguiente:

$$\text{dificultadAcierto} = \frac{1}{d \times c}$$

donde d representa la distancia y c el número de colisiones con planetas.

5. Conclusiones

Se ha ejecutado el algoritmo con ambas funciones de aptitud durante 1000 iteraciones en el caso de la función de las intersecciones y 500 en el de minimizar la atracción en el planeta objetivo (valores totalmente factibles para la generación del contenido de forma online), para generar niveles con la intención de probarlos en jugadores reales. Con cada función de aptitud se ha tomado el mejor individuo de cada una de las tres clases de la población estructurada. El fitness obtenido por los tres niveles del método de las intersecciones se muestra en el cuadro 1, mientras que los de la otra función están en el cuadro número 2.

Nivel	Fitness	Frec. de aciertos	Dificultad simulador
Difícil	418.96	0.065 %	0.144
Medio	391.51	0.457 %	1.056
Fácil	354.25	3.162 %	6.503

Cuadro 1. Intersecciones

Nivel	Fitness	Frec. de aciertos	Dificultad simulador
Difícil	2.2238	1.697 %	3.655
Medio	2.3128	0.443 %	2.286
Fácil	2.3664	4.242 %	22.003

Cuadro 2. Atracción mínima

En resumen, en esta memoria se ha mostrado un juego sencillo, basado en una idea innovadora que define de forma implícita (aunque también básica) un motor de físicas. El algoritmo dise nado para la generación de contenido introduce la idea de utilizar poblaciones estructuradas para ofrecer a los jugadores un amplio espectro de niveles de distinta dificultad, adaptado a todo tipo de jugadores, con la idea no resultar aburrido a los más experimentados ni frustrante a aquellos con menor habilidad. Para ello se han definido dos funciones de aptitud que pueden ejecutarse de forma online, con lo que se podría generar un juego infinito adaptado a cada tipo de jugador. La calidad de las soluciones del algoritmo evolutivo con las funciones de aptitud implementadas han sido contrastadas por medio de otra función de evaluación de simulación directa. Esta función, aunque no está pensada para ser usada de forma online por su elevado coste, si que podría usarse para evaluar el nivel que vaya a ofrecerse al jugador. Así pueden evitarse situaciones como que se plantee un nivel de imposible resolución al jugador, algo que no resultaría aceptable. La sencillez del juego permite que este mismo algoritmo pueda ser usado en otros juegos para distribuir por su mapa cualquier

tipo de contenido opcional, con solo definir otras funciones de aptitud concretas. Resulta de especial interés el algoritmo de cruce implementado, que resulta más general que el concepto de recombinación en un punto para este juego.

Agradecimientos Este trabajo está parcialmente financiado por la Junta de Andalucía dentro del proyecto P10-TIC-6083 (DNEMESIS) y la Universidad de Málaga. Campus de Excelencia Internacional Andalucía Tech.

Referencias

1. Total consumer spent on all games content in the U.S. estimated between 16.3 to 16.6 billion
2. Aarseth, S.J., Aarseth, S.J.: Gravitational N-Body Simulations: Tools and Algorithms. Cambridge University Press (2003)
3. Esparcia-Alcázar, A.I.: Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings. Springer (2013)
4. Frade, M., de Vega, F.F., Cotta, C.: Evolution of artificial terrains for video games based on accessibility. In: Applications of Evolutionary Computation, pp. 90–99. Springer (2010)
5. Frade, M., de Vega, F.F., Cotta, C.: Evolution of artificial terrains for video games based on obstacles edge length. In: Evolutionary Computation (CEC), 2010 IEEE Congress on. pp. 1–8. IEEE (2010)
6. Hastings, E.J., Guha, R.K., Stanley, K.O.: Automatic content generation in the Galactic Arms Race video game. Computational Intelligence and AI in Games, IEEE Transactions on 1(4), 245–263 (2009)
7. Hastings, E.J., Guha, R.K., Stanley, K.O.: Evolving content in the Galactic Arms Race video game. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on. pp. 241–248. IEEE (2009)
8. Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A.: Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 9(1), 1 (2013)
9. Hom, V., Marks, J.: Automatic design of balanced board games. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). pp. 25–30 (2007)
10. Lucas, S.M.: Computational intelligence and ai in games: a new ieee transactions. Computational Intelligence and AI in Games, IEEE Transactions on 1(1), 1–3 (2009)
11. Lucas, S.M., Mateas, M., Preuss, M., Spronck, P., Togelius, J., Cowling, P.I., Buro, M., Bida, M., Botea, A., Bouzy, B., et al.: Artificial and Computational Intelligence in Games. Schloss Dagstuhl-Leibniz-Zentrum für Informatik GmbH (2013)
12. Man, K.F., TANG, K.S., Kwong, S.: GENETIC ALGORITHMS.: Concepts and Designs, Avec disquette, vol. 1. Springer (1999)
13. Pedersen, C., Togelius, J., Yannakakis, G.N.: Modeling player experience in super mario bros. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on. pp. 132–139. IEEE (2009)
14. Shaker, N., Yannakakis, G.N., Togelius, J.: Towards automatic personalized content generation for platform games. In: AIIDE (2010)

15. Togelius, J., De Nardi, R., Lucas, S.M.: Towards automatic personalised content creation for racing games. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on. pp. 252–259. IEEE (2007)
16. Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelback, J., Yannakakis, G.N.: Multiobjective exploration of the Starcraft map space. In: Computational Intelligence and Games (CIG), 2010 IEEE Symposium on. pp. 265–272. IEEE (2010)
17. Togelius, J., Preuss, M., Yannakakis, G.N.: Towards multiobjective procedural map generation. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games. p. 3. ACM (2010)