ACTAS DE

SEED 2013

PRIMER SIMPOSIO ESPAÑOL DE ENTRETENIMIENTO DIGITAL

UNIVERSIDAD COMPLUTENSE DE MADRID

18 Y 19 DE SEPTIEMBRE DE 2013

SEED @ CEDI 2013

El primer foro científico para el intercambio de ideas y resultados de investigación sobre el diseño, la ingeniería y la teoría de la tecnología aplicada al entretenimiento en España. Trabajo en inteligencia artificial, informática gráfica, ingeniería del software e interacción persona-computador aplicado a la creación de sistemas de entretenimiento digital.

Pedro Antonio González Calero y
Marco Antonio Gómez Martín (Eds.)

Actas del Primer Simposio Español de Entretenimiento Digital

Universidad Complutense de Madrid,
18 y 19 de Septiembre, 2013

Editores

Pedro Antonio González Calero
Departamento de Ingeniería del Software e Inteligencia Artificial,
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, España
E-mail: pedro@fdi.ucm.es

Marco Antonio Gómez Martín
Departamento de Ingeniería del Software e Inteligencia Artificial,
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, España
E-mail: marcoa@fdi.ucm.es

ISBN: 978-84-695-8350-0

Prefacio

El Simposio Español de Entretenimiento Digital, SEED 2013, nace con el objetivo de convertirse en el primer foro científico para el intercambio de ideas y resultados sobre el diseño, la ingeniería y la teoría de la tecnología aplicada al entretenimiento en España. Esto incluye, por una parte, trabajo en inteligencia artificial, informática gráfica, ingeniería del software o interacción persona-computador aplicados a la creación de sistemas de entretenimiento digital, y, por otra, resultados de la aplicación de las tecnologías propias del entretenimiento digital a la enseñanza, la medicina, la comunicación o el arte.

En su primera edición, este Simposio se celebra en el marco del Congreso Español de Informática (CEDI 2013) lo que, además de facilitar todos los aspectos prácticos de la organización, ha de permitir la búsqueda de sinergias con otras comunidades científicas del ámbito Informático.

SEED nace con la vocación de facilitar la transferencia de resultados de investigación a la industria y por ello se ha promovido la participación de profesionales y se ha contado con figuras relevantes de la industria nacional del videojuego como ponentes invitados.

Agosto 2013

Pedro Antonio González Calero
Marco Antonio Gómez Martín

Program Committee

Laura Baigorri Ballarin	Universidad de Barcelona
Isabel Barbancho	Universidad de Málaga
David Camacho	Universidad Autónoma de Madrid
Miguel Chover	Universidad Jaume I
Antonio J. Fernández Leiva	Universidad de Málaga
Pablo Alejandro Figueroa Forero	Universidad de los Andes (Bogotá)
Pascual González	Universidad de Castilla La Mancha
Pedro González Calero	Universidad Complutense de Madrid
Marco Antonio Gómez Martín	Universidad Complutense de Madrid
Javier Jaén	Universidad Politécnica de Valencia
Sergi Jorda	Universidad Pompeu Fabra
Moisés Mañas	Universidad Politécnica de Valencia
José Pascual Molina Massó	Universidad de Castilla La Mancha
Antonio M. Mora García	Universidad de Granada
Jaime Munárriz	Universidad Complutense de Madrid
Mateu Sbert	Universidad de Gerona

Table of Contents

Additional Reviewers

Fernández de Vega, Francisco
Mocholi, Jose A.

Modelling Human Expert Behaviour in an Unreal Tournament™ 2004 Bot

A.M. Mora, F. Aisa, R. Caballero, P. García-Sánchez,
P.A. Castillo, J.J. Merelo, and P. De las Cuevas

Departamento de Arquitectura y Tecnología de Computadores.
Universidad de Granada (Spain)
{amorag,pgarcia,pedro,jmerelo,paloma}@geneura.ugr.es,
francisco_aisa@hotmail.com, rcabamo@gmail.com

Abstract. This paper presents a deep description of the design of an autonomous agent (bot) for playing 1 vs. 1 dead match mode in the first person shooter Unreal Tournament™ 2004 (UT2K4). The bot models most of the behaviour (actions and tricks) of an expert human player in this mode, who has participated in international UT2K4 championships. The Artificial Intelligence engine is based on two levels of states, and it relies on an auxiliary database for learning about the fighting arena. Thus, it will store weapons and items locations once the player has discovered them, as a human player could do. This so-called expert bot yields excellent results, beating the game default bots in the hardest difficulty, and even being a very hard opponent for the human players (including the expert).

1 Introduction

Autonomous agents in videogames (so-called *bots*) have tried to behave as human players from their emergence in the first years of the nineties. They normally model a part of a human expert knowledge regarding the game, in order to become a competitive opponent to play against some other humans or bots, or lately, to cooperate with or aid the main players. They are also named non-playing characters (NPCs), classical secondary characters in conversational or rol games, but referring nowadays to the same concept as bots: an Artificial Intelligence (AI) engine which is able to perform the same actions in a game as a human player.

First Person Shooter games (FPSs) are one of the most profitable area in the study and implementation of bots. In these games the player can only see the hands and the current weapon of her character, and has to fight against enemies normally by shooting to them. They usually offer a multiplayer fighting mode placed in a limited arena. From their first years (begin of the nineties) these games have been open to the creation of bots, initially aimed to their graphical aspect, but later giving tools to implement even their whole AI (or a part).

Unreal™ , launched for PC by Epic Games in 1998, had a great success since it incorporates the best multiplayer mode to date. In addition it started an open-code philosophy to make it easier the game-modding (modification), including

with every copy of the game an editor (UnrealEd), an own programming language (UnrealScript), and a compiler, to add or change almost whatever the user desires: scenarios, items, characters, etc. The AI engine is also open, so the state-based behaviour implemented in the game characters can be changed in a number of ways, just having some critical actions restricted.

Moreover, some additional tools have arisen a few years ago, such as GameBots [1], a *mod* (new utility for the game) that allows the control of characters (bots) in the game through network connections to other programs. The game sends character's sensory information to the client program, which can decide the actions the bot will take. These actions are sent back to the game which interprets them for the bot movement, shooting, jumping, etc. It was initially released for the Unreal sequel *Unreal Tournament™ (UT)* and later implemented for *UT 2003*. Over the basis of that tool, it was later launched *Pogamut* [2], which defines an interface (using GameBots architecture) to program the bots externally using Java. It was implemented for Unreal Tournament™ 2004 (UT2K4).

These open-code possibilities and external tools are the main reasons why this game environment have been widely considered in the computational intelligence researching field [3–9].

This work is also embedded in the UT2K4 environment, and uses Pogamut for implementing a bot which we have baptised as *UT Expert Bot (E-Bot)*. Its behaviour (AI) is shaped as a Finite State Machine (FSM) [10] (based on two levels of states), that describes a complex set of rules. These rules are based on the knowledge of an UT2K4 Spanish expert player. It has been modelled according to his experience, including several tricks, as the humans players do, to achieve a better behaviour in the game.

2 Unreal Tournament™ 2004 Game Features

As previously commented, Unreal™ was a very famous FPS published in 1998 for PCs. It presented a very good single mode, but the multiplayer possibilities gave it (and still give) a great success. Thus, in 1999 it was released Unreal Tournament™, which turned the series into multiplayer-aimed games, having several arenas specifically designed for massive battles between humans and/or bots. The most successful sequel in this series was Unreal Tournament 2004 (UT2K4), due to the original weapons, the excellent design of scenarios and the famous amazing opponents' AI, based on states and events, inside a huge Finite State Machine [10] (where plenty of *states* and *substates* are present), and including several *optimised scripts* (predefined actions).

This framework inherited all the features that the first Unreal Engine offered for modders (people who add or change components in the game), such as maps, weapons, items, or even bots. Some of these features are:

– It includes a proprietary programming language, called *UnrealScript*, which combines the C and Java syntax, with some useful features, such as a garbage collector. It is object-oriented and handles implicit references (every object is represented by a pointer).

– This language includes the declaration and handling of *states*, which are the most powerful feature of the Unreal bots' AI. They model a bot status and behaviour at a time, and are defined as classes. During the game (depending on the bot location and status), the current state of the bot is changed, and the functions defined in it are performed.
– In addition to the game, a programming environment, named *UnrealEditor* is included. It makes it easier the management of the hierarchy of classes, as well as the creation of new classes which inherit from the existing ones.
– It is possible to change an existing class (making a mod) by creating another one which inherits from it, and modifying the code of the desired functions or methods inside the new class.

In addition, this new engine (Unreal Engine 2.5) fixed some of the drawbacks that UnrealScript had, such as the small number of elements in arrays, the limitations in the number of iterations in loops, or the file input/output.

Moreover there were some projects which created specific tools for interacting with UT and UT2K4 engines, such as the aforementioned Gamebots [1] and Pogamut [2]. These projects let the user to implement mods (mainly bots), using more powerful and flexible programming languages than the limited UnrealScript, like Java. The latter is the tool we have considered in this work due to its 'simplicity' and proved value in the implementation of bots (it has been widely used by several authors in the literature).

Going back to the game, the most traditional combat mode is *Death Match*, in which the player must eliminate as many enemies as possible before the match ends, avoiding being defeated by other players. Everyone has a limited number of health points, which are decreased as the character gets hurt. If the health counter goes down to 0, the player is defeated, and a *frag* is added to the last player who shot him. After being killed, the player is then respawned (usually in the same place or quite near) in the scenario. A match ends when the termination conditions (typically a limit of frags/kills or time) are reached.

In order to aid players to succeed in the match, some elements appear periodically in the arena: *weapons* (with limited ammunition, and an associated power) to defeat the enemies , and *items*, that provides the player with some advantages, such as extra health, high jump, invisibility or ammunition, for instance.

Dead Match mode is usually played by a number of players/bots up to 32 in the recent games of the series, but there is a very interesting variation; the *1 vs 1 Death Match* battle. It is considered in the official competitions of UT2K4 for human players and presents some specific rules and constraints: there are some items forbidden (U-Damage), weapons are not respawned, and the match runs for 15 minutes, not for a number of frags (number of enemies killed).

3 State of the Art

In the nineties, bots started to be widely used in FPSs, when Quake™ became the most successful game in including user-created autonomous characters. It presented not only the option of playing against machine-controlled bots, but

also the possibility to modify them (just in appearance or in a few other aspects), or create new ones by means of a programming language named QuakeC, which unfortunately was strictly limited and hard-constrained.

Unreal™ series appeared some years later, being (as aforementioned) the first game in including an easy programming environment and a more powerful language, thus plenty of bots were developed (just a few applying metaheuristics or complex AI techniques), and most of these bots were based on predefined hard-coded scripts.

One of the most fertile areas inside computer games and AI, is devoted to get improvements on some components of the characters' AI. These studies appeared more than a decade ago We started our research in this field in 2001, publishing our results in national conferences [3] (in Spanish). We applied a Genetic Algorithm to improve the parameters in the bots AI core (as later other authors did [11]), and to change the way of controlling the bots by automatically redefining, by means of Genetic Programming (GP), the standard set of rules of the main states. Several other evolutionary approaches have been published, such as [12], where evolution and co-evolution techniques have been applied, [7] which applies an evolutionary rule-based system, or the multi-objective approach presented in [9] which evolves different specialised bots. The two latter have been developed inside the Unreal Tournament™ 2004 environment (UT2K4).

Also in the first years, studies involving other techniques arose, such as [13], where the authors used self-organizing maps and multilayer perceptrons, or as [14], which applied machine learning, to achieve in both cases human-like behaviour and strategies, in Quake™ 2 and 3 games respectively. Recent studies related to computational intelligence (a branch of the AI) , are based on bots controlled by neural networks (NNs). For instance, the authors in [15] train NNs by reinforcement learning, and Schrum et al. [16] evolve NNs searching for multimodal behaviour in bots.

The design of human-like bots (try to imitate humans' behaviour) is an interesting research line which has become popular a few years ago in the FPS scope, and specifically inside UT2K4 series due to the international Botprize competition [17], which searches for the most human' bot. Several papers have been published regarding this area, such as the work by Soni and Hingston [6], in which the authors use a NN to train a bot, based in recorded data of human plays, for playing UT2K4 as a human. Schrum et al. [18] applied a similar strategy, relying in human records of matches in the same game to support the multiobjective evolution of a NN. This is evolved to get the best performance and skilled actions, but the bot humanity is corrected, mainly regarding the navigation capability, by means of the human data. There are other approaches such as [19] which is based in reinforcement learning and self-organizing NN, or [20] in which the authors model a part of the brain workspace and neural interactions (through NN).

Our work is enclosed in this scope, however it presents the design of a human-like bot created by modelling its AI relying in the knowledge of a human expert player. It presents a novel state-based approach, which considers main and sec-

ondary states, and it also deeply describes the AI engine workflow, and the realistic (from the human point of view) use of a database for learning' the arenas.

4 UT2K4 Expert Bot

The design of the expert bot (*E-Bot*) AI is based on the knowledge of an expert (human) player, *Me$$!@h*, who belongs to the best Spanish UT2K4 clan, *Trauma Reactor* (*dRâ*), and has participated in some European championships. He is one of the authors of the paper and has designed and implemented the main work flow of the bot's behaviour. As a summary, *E-Bot* is implemented considering hierarchical states (primary and secondary), a set of rules to decide the correspondent states, and a (simple) database. In the following sections all these terms will be described, starting with a summarised expert analysis of the main elements in the game.

It is important to point out that we have considered the rules of the official championships then, as stated, weapons are not respawned, there are some forbidden items, and there is a time limit per match instead a number of frags limit.

4.1 Expert knowledge

The items and weapons are critically valuable in a *1 vs 1 Death Match*, since they could mean the difference between win or lose against an opponent. It is even more important to consider the respawn *timing* of each of them, i.e. the time it takes to appear again once an item has been picked (weapons are not respawned). The importance grows in this mode since if one player picks one item or weapon, she prevents the opponent for profiting it (while she does, of course). Expert players must control this time in order to move to a known respawn area in the moment an item appears. These conditions (timing and locations) depend on the map, which the player should know in advance.

The following list describes the main *items* to consider:

- *Super Shield*: the most important item in 1 vs 1 matches. It gives the player 100 shield points (the maximum is 150).
- *Small Shield*: similar to previous one but it gives 50 shield points.
- *Health Pack*: gives 25 health points. It just can be used if the player's health is lower than 100.
- *Health Vial*: gives 5 health points. This can be always used until the player reaches 199 health points.
- *Adrenaline*: gives 5 adrenaline points. If the player reaches 100 of these points, she can obtain a reward in a limited time, such as:
  - Invisibility: the player cannot be seen.
  - Berserker: faster shots.
  - Speed: faster movement.

- Booster: player's health and shield is regenerated.

– *Ammo*: gives ammunition for a specific weapon. There is a different item per weapon.

Weapons in UT2K4 have been carefully' designed to be all of them equally useful and important (as a difference to other games), so there is no one absolutely better than the rest. Every weapon has a perfect moment to be used and take advantage over other weapons. The choice of this moment is an expert decision that depends on several factors such as the player's and enemy's health, the distance to enemy or the height/level where she is placed. The list of weapons along with their utility is summarised in Table 1.

Table 1. UT2K4 Weapons.

	Useful for/when	*Main shot*	*Secondary shot*
Shield Gun	retreat and protection, player's health low	can damage enemy, impulse for jump	shield for reducing the received damage
Assault Rifle	hidden enemy known	fire shot, low damage	parabolic launch of grenade
Link Gun	enemy tries to escape, covering an area	energy balls in straight direction	links the enemy to the ground
Bio Rifle	surprising the enemy, enemy's health is higher than ours	viscous balls which glue to floor and walls, lightly damage to enemy who moves close to them	charge, big ball which could kill the enemy if he is close
Minigun	enemy's health is low, medium/close distance	very fast small bullets	slow big bullets
Rocket Launcher	medium distance	one rocket, high damage on impact	several rockets, spam shots
Flak Cannon	enemy is on different level	burst of bullets, close distance	parabolic ball
Shock Rifle	all distances, very versatile	energy laser	energy ball which can explode, high damage
Lightning Gun	far distances	sniper rifle	sniper sight

The *movement* is another key factor in UT2K4, since it could mean the difference between win or lose. Players usually move jumping, in order to avoid being shot easily. This is a problem of our approach because the Pogamut movement module does not implement the jump.

4.2 Database

When a human player starts playing in a map unknown to him, it is usual to take some turns to investigate/analyse the scenario and learn' where the important things are, i.e. mainly weapons and items, but also advantageous positions and hiding places.

This behaviour has been implemented in the expert bot by means of a simple database. It features a single table, which has as fields:

– *Unreal_ID*: unique identification of an item in Pogamut.
– *Type*: health, shield, ammo, or weapon, among others.
– *Name*: for instance Flak Cannon, Small Shield or Mini Health.
– *Map*: name of the map where the item is placed.

The initial idea was to include lots of information, such as respawn points, usual paths, heat zones, last plays or moves, which will imply a high level of knowledge for the bot, but also a high computational cost when the it would have to interpret and use all this data. Since the bot must react in real-time, we decided (for this initial version) to store just the most important and useful information. To this end we performed several test matches using different combinations of these data. Finally we concluded that the most relevant data to record and recover later were the described above.

The most important field is *Unreal_ID*, because Pogamut offers information about it, including its location. It is important to note that the table is filled with the information about the items that the bot discovers while it is moving across the map (searching for enemies or just exploring to learn). There are no records about items not being seen by the bot, although it could be possible in Pogamut (this would be a trick).

This database has been designed and implemented using SQLite, due to its light weight', simplicity and portability. It just consists in a file which we can move together with the bot code from one PC to another one. It will be extended in future versions.

4.3 AI work flow

The use of Pogamut implies we cannot consider the FSM defined in the UT2K4 bot's AI, thus, a complete AI engine must be designed and implemented. Pogamut offers some high-level functions which we have profited, such as basic navigation from point to point in a map. As commented in the introduction section, Pogamut offers to the user sensory information that the bot perceives during the match. To this end the bot can implement the so-called *listeners*, which are triggers that the programmer defines in advance; when an event related to this trigger happens, the bot performs the associated actions.

The main idea in *E-Bot* AI module is the consideration of two levels of states: primary and secondary. The first ones correspond to general actions to perform, while the latter are devoted to perform additional tasks within the main action flow. For instance, the bot can decide to attack the enemy, but if its health is low also search for any health item.

The *primary states* descriptions are:

- *Attack*: the enemy is in sight. The bot moves towards him, shooting with its best weapon and avoiding opponent's shots. Otherwise, the bot moves side to side (in a pendular movement).
- *Hunt*: the bot tries to hunt the enemy. If he is in sight, it pursues him; otherwise the bot infers its position considering the noise that the enemy does.
- *Retreat*: the bot tries to escape from the enemy. If he is in sight, the bot moves facing him, avoiding his shots and using the shield weapon (if possible). Otherwise the bot shoots at him while it escapes. If the enemy's position is not known, the bot infers it hearing at the noises he produces and moves far from them.

– *Greedy*: the bot moves around the map picking the items it needs. The bot knows where they are if it has been previously in the map and has stored its position in the database. If the enemy is in sight, the bot tries to defend itself while it picks the items.
– *Camp*: the bot waits for the enemy in a static position.

The *secondary states* are only devoted to change the movement of the bot:

– *Defensive Profile*: the bot tries to escape from the enemy.
– *Offensive Profile*: the bot moves towards the enemy.
– *Pickup Weapon*: the bot picks all the weapons it founds.
– *Pickup Ammo*: the bot picks all the ammunition it founds.
– *Pickup Health*: the bot picks all the health items it founds.
– *Critical Health*: the bot moves towards the closest health item it knows (using the database).
– *Critical Weaponry*: the bot moves towards the closest weapon it knows (using the database).

The choice of the primary state is performed by the AI engine following the flow chart shown in Figure 1. The *Camp* state is not included because it is

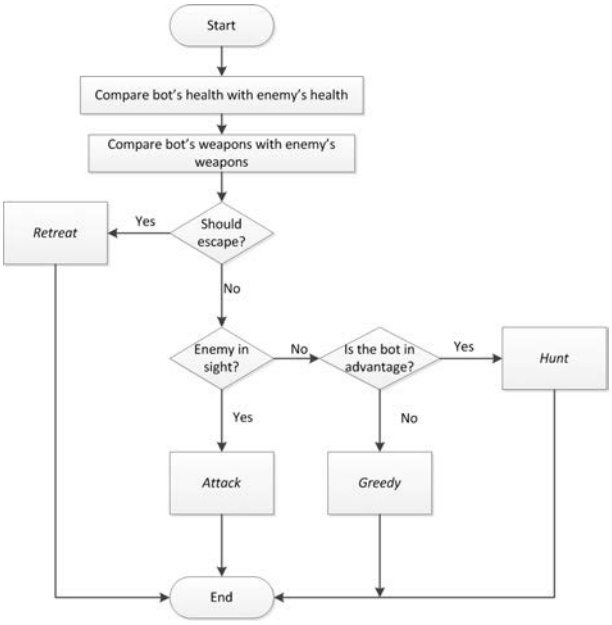


Fig. 1. Primary States selection flow chart

a special state with very particular conditions. The comparisons and decisions required to choose the state are quite complex, because they consider several factors and situations that the human expert knows. For instance, the comparison between weapons depends on several parameters and weights, because of the

existent balance between the power and usefulness of all of them, which could mean that a weapon is the best in a specific situation (different levels, hidden opponent, close or far enemy), but it could be the worse on a different position.

The AI engine (composed by a huge system of rules) chooses the primary state at a time, but while the bot is performing the actions associated to it, the engine continues checking conditions (for instance the timing of every item), receiving sensory information, or checking the bot status, for instance. This way, a secondary state can also be set. However the engine can stop and change, at any time, both the primary and/or the secondary state, depending on the conditions, received information and status, giving an extra flexibility' level which a FSM usually do not offer.

As a general summary, the bot tends to be defensive if its health is lower than the enemy's, unless the latter has a critical health level, so the bot will attack him to win a frag. In similar health conditions the attitude depends on the weaponry (in comparison with the enemy's weapons). If the bot's health is good, it is quite aggressive. However, there are several conditions and situations considered which could change the states and actions to perform. For example, the difference in height (levels). The *E-Bot* general flow chart can be seen in Figure 2.

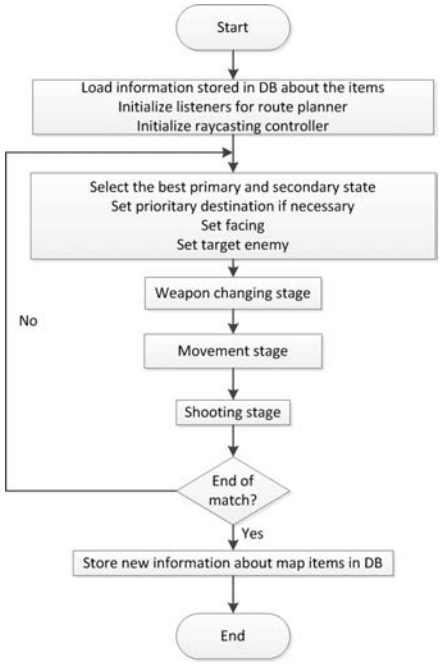


Fig. 2. *E-Bot* general flow chart. The route planner controls the navigation around the map, while the raycasting controller traces vision lines for avoiding obstacles and seeing enemies.

The source code of *E-Bot* is available under a GPL license at `https://github.com/franaisa/ExpertAgent`.

5 *E-Bot* Results

In order to test the expert bot some experiments has been conducted. The first consists in launching four different game matches in the *1 vs 1 - Death Match* mode, having as players a game standard bot in the hardest difficulty (*StdBot*), and our expert bot (*E-Bot*).

Two different arenas have been considered, namely *DM-Idoma* and *DM-Ironic*, since they are two of the most famous maps, used in competitions and with a design that offers almost any possible fighting situation: different levels, hidden zones, viewpoints and camping areas, etc.

The bots have been fighting following the commented championship rules (as *E-Bot* was designed) along 15 minutes. The results of every match as well as the average in each map are shown in Table 2.

Table 2. Scores (number of frags) of the test fights between *E-Bot* and a Standard UT2K4 bot in the hardest difficulty (*StdBot*).

Map	*E-Bot*	*StdBot*
DM-Ironic	17	6
	22	8
	19	9
Average	19.33	7.67

Map	*E-Bot*	*StdBot*
DM-Idoma	23	14
	20	9
	25	8
Average	22.67	10.33

As it can be seen *E-Bot* outperforms the standard UT2K4 bot in all the matches, even in the hardest difficulty level, which is a true challenge for a medium level human player. This leads us to conclude that the expert bot has been successfully designed and implemented.

As an additional test, our expert human player fought some matches against *E-Bot*, always beating it. It is an expected result, since this player is one of the best in Spain. Thus, we have also proved the value of the expert bot versus four medium level human players (they play frequently but they are non-professional'). One of them lost all the matches against the bot, and the rest had serious difficulties to beat *E-Bot*, yielding a considerable amount of frags in the matches, so it could be considered as high-level enemy.

6 Conclusions and Future Work

In this work, the design of a human-like bot for the First Person Shooter game Unreal Tournament™ 2004 (UT2K4) has been described. It models the expert knowledge of a Spanish high-level player so, its has been named Expert-Bot (*E-Bot*). Its behaviour is shapes as a finite state machine with two levels of states (primary and secondary), each of them with an associated priority for performing actions. This work flow is flexible, so the AI engine can alter or change

it depending on the conditions and on the bot's or enemy's status. Moreover, the bot uses a database for learning' the maps, as a human player would do the first time she navigates around a new arena. *E-bot* has been designed for fighting in 1 vs 1 death match mode, considering the official competition rules.

Some experiments have been conducted, firstly proving that *E-Bot* outperforms the standard UT2K4 bots, even in the hardest difficulty level. Then, a set of matches against medium-level human players have been performed with quite good results: one human has been beaten and the rest have had high difficulties (several frags) to win. The authors consider these results as a success, however there still remain several points of improvement to create a high competitive bot. According an expert player's opinion (one of the authors), *E-Bot* does not perform as well as expected (from the behavioural point of view), so it would be necessary to improve its behaviour rather than its performance in results.

This way, the first line of research could be the improvement of the bot in its main flaw, the movement, since the implementation which offers Pogamut does not include jumps, which in fact are basic in an expert control. Thus, the possibility of jumping must be included in its actions, in order to model a harder opponent (more difficult to kill at least).

The database complexity could be increased for considering other important information for the bots, such as respawn points, heat zones, better paths, navigation points. But some additional work should be done in order to deal with this data in real-time, avoiding a delay in the performance of the bots.

Once the behaviour would be improved, then several researching could be made with regard to the finite state machines improvement (set of behavioural states). Following other approaches by the authors, evolutionary computation algorithms [3, 8] could be used to improve both, the set of behavioural rules and the set of threshold parameters which, in term, determine which rules are triggered and so, define the bot's behaviour.

Finally, another research line could be the adaptation or optimisation of *E-Bot* for multiplayer and team modes, which are so far the most famous among the players.

Acknowledgements

This paper has been funded in part by projects P08-TIC-03903 (Andalusian Regional Government), TIN2011-28627-C04-02 (Spanish Ministry of Science and Innovation), and project 83 (CANUBE) awarded by the CEI-BioTIC UGR.

References

1. WEB: (Gamebots project) http://gamebots.sourceforge.net/.
2. WEB: (Pogamut 3 project) http://pogamut.cuni.cz/main/tiki-index.php.
3. Montoya, R., Mora, A., Merelo, J.J.: Evolución nativa de personajes de juegos de ordenador. In et al., E.A., ed.: Actas primer congreso español algoritmos evolutivos, AEB'02, Universidad de Extremadura (2002) 212–219

4. Jacobs, S., Ferrein, A., Lakemeyer, G.: Controlling unreal tournament 2004 bots with the logic-based action language golog. In: Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference. (2005)
5. Karpov, I., D'Silva, T., Varrichio, C., Stanley, K.O., Miikkulainen, R.: Integration and evaluation of exploration-based learning in games. In Louis, S.J., Kendall, G., eds.: CIG, IEEE (2006) 39–44
6. Soni, B., Hingston, P.: Bots trained to play like a human are more fun. In: IEEE International Joint Conference on Neural Networks, IJCNN'08. (2008) 363–369
7. Small, R., Bates-Congdon, C.: Agent Smith: Towards an evolutionary rule-based agent for interactive dynamic games. In: IEEE Congress on Evolutionary Computation 2009 (CEC'09). (2009) 660–666
8. Mora, A.M., Montoya, R., Merelo, J.J., García-Sánchez, P., Castillo, P.A., Laredo, J.L.J., Martínez, A., Esparcia, A.I.: Evolving bot AI in Unreal. In et al., C.D.C., ed.: Applications of Evolutionary Computing, Part I. Volume 6024 of LNCS., Springer (2010) 170–179
9. Esparcia-Alcázar, A.I., Martínez-García, A., Mora, A.M., Merelo, J.J., García-Sánchez, P.: Genetic evolution of fuzzy finite state machines to control bots in a first-person shooter game. In Pelikan, M., Branke, J., eds.: GECCO, ACM (2010) 829–830
10. Booth, T.L.: Sequential Machines and Automata Theory. 1st edn. John Wiley and Sons, Inc., New York (1967)
11. Cole, N., Louis, S.J., Miles, C.: Using a genetic algorithm to tune first-person shooter bots. In: Proceedings of the IEEE Congress on Evolutionary Computation 2004. (2004) 139–145
12. Priesterjahn, S., Kramer, O., Weimer, A., Goebels, A.: Evolution of human-competitive agents in modern computer games. In: IEEE World Congress on Computational Intelligence 2006 (WCCI'06). (2006) 777–784
13. Thurau, C., Bauckhage, C., Sagerer, G.: Combining self organizing maps and multilayer perceptrons to learn bot-behaviour for a commercial game. In Mehdi, Q.H., Gough, N.E., Natkine, S., eds.: GAME-ON, EUROSIS (2003) 119–123
14. Zanetti, S., Rhalibi, A.E.: Machine learning techniques for FPS in Q3. In: ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology, New York, NY, USA, ACM (2004) 239–244
15. Cho, B.H., Jung, S.H., Seong, Y.R., Oh, H.R.: Exploiting intelligence in fighting action games using neural networks. IEICE - Trans. Inf. Syst. **E89-D**(3) (2006) 1249–1256
16. Schrum, J., Miikkulainen, R.: Evolving multi-modal behavior in NPCs. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium On. (2009) 325–332
17. 2K-Games: The 2k botprize competition (2012) http://www.botprize.org.
18. Schrum, J., Karpov, I.V., Miikkulainen, R.: Ut$\hat{2}$: Human-like behavior via neuroevolution of combat behavior and replay of human traces. In: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2011), Seoul, South Korea, IEEE (2011) 329–336
19. Wang, D., Subagdja, B., Tan, A.H., Ng, G.W.: Creating human-like autonomous players in real-time first person shooter computer games. In: Proceedings of the 21st Annual Conference on Innovative Applications of Artificial Intelligence (IAAI'09), Pasadena, USA (2009) 173–178
20. Fountas, Z., Gamez, D., Fidjeland, A.: A neuronal global workspace for human-like control of a computer game character. In Cho, S.B., Lucas, S.M., Hingston, P., eds.: CIG, IEEE (2011) 350–357

Domain Modeling as a Contract between Game Designers and Programmers *

David Llansó, Pedro Pablo Gómez-Martín,
Marco Antonio Gómez-Martín and Pedro A. González-Calero

Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
email: {llanso,pedrop,marcoa,pedro}@fdi.ucm.es

Abstract. Collaboration between programmers and designers is a crucial problem in game development. Designers decide what need to be done and programmers decide what can be done, and they need to collaborate in order to produce the best possible experience for the player. In this paper we present a methodology that alleviates this problem by putting a declarative model of the game domain as the contract between programmers and designers.

1 Introduction

Game development is nearly an art, where new ideas must be completely implemented and tested to be sure that they are fun to play. Trial and error of new game concepts is needed, what requires an elaborated software architecture that provides the needed flexibility and change tolerance. In addition, game development requires the collaboration of artists that picture the world, programmers that make the world work, and designers that create a game out of the visuals and dynamics of the virtual world. Collaboration between programmers and designers is specially important and, at the same time, specially difficult. Designers decide what need to be done and programmers decide what can be done. A fluent and constant communication between designers and programmers is needed, where both know the needs and objectives of the others.

In order to facilitate the collaboration between programmers and designers in game development, we propose a computer-aided methodology that involves both designers and programmers, and is based on two main ideas: 1. put a declarative model of the game domain as the contract between programmers and designers, and let designers formalize the elements of the game that programmers must develop, and 2. use a component-based software architecture, instantiated as an elaboration of the game domain that connects the declarative model with actual code.

With the purpose of supporting such methodology we have developed an authoring tool, *Rosette*, which supports the authoring of the game domain model and the elaboration of source code components from the model. In this paper we describe the methodology in next Section, before going into a brief description of the tool in Section 3. Section 4 presents an example game created with the proposed methodology, and finally, Section 5 concludes the paper.

* Supported by the Spanish Ministry of Science and Education (TIN2009-13692-C03-03)

2 Methodology

As a high-level description, the methodology is split in three phases:

- Designers define the *game domain* from the gameplay point of view, using a rich knowledge representation.
- Programmers enrich that information adding high-level implementation aspects that designers could not be aware of.
- Programmers implement all the game functionality in some programming language and designers distribute entities through the level editor and create character behaviours.

Prior to detail each phase, please note that our methodology supports these phases being reiterated as needed, but this feature is out of the scope of this paper; we will focus in just one iteration and, for simplicity, we will assume that the game is developed using a waterfall model.

Game creation begins with designers defining the complete game functionality. Gameplay is provided by *entities* (players, enemies, items, and so on) and designers must agree on all the entity types the game will have and specify them in an ontological (hierarchical) way.

Entities have a state (position, health, armor, ...) and a set of functionalities (actions they can do) that must be also specified. Designers provide all this information creating the complete *game domain* that constitutes a semantic specification of the gameplay and becomes a *contract* between designers and programmers. If any decision endangers the implementation of the game, programmers must report it before accept this game domain.

Once they come to an agreement, programmers are in charge of breath life into all those entities designers have envisioned. They will start using the semantic specification of the game domain, *adding* new entities, attributes (state) and actions that, although go unnoticed from a designer point of view, will be important from the implementation perspective. Entities are then split into software components that will contain subsets of the state and functionality that the starting entities had. Component distribution can be hand-made, or even created using a supporting tool and then fine-tuned as desired.

Once programmers have completely modelled entities and components, the semantic model of the game domain is used to create source code scaffolding for all of them, adapted to the target platform and underlying technologies. Placeholders mark the specific points where programmers must write code to implement the components behaviour. Therefore, they do not have to write all the game logic and component management layer, but just the code that differs between components. All the code that the target platform (say Unity3D or a in-house game engine) requires to manage the component (usually constructors, attribute declarations, and so on) are machine generated from the game domain.

Similarly, the semantic model is also used to create files that feed the tools that game levels designers use to create them. Prefabs (Unity3D) or blueprints/archetypes information (C++ engines) are generated and become the concrete entity definitions that can be put into the game levels. They can also be used for the behaviour definition where

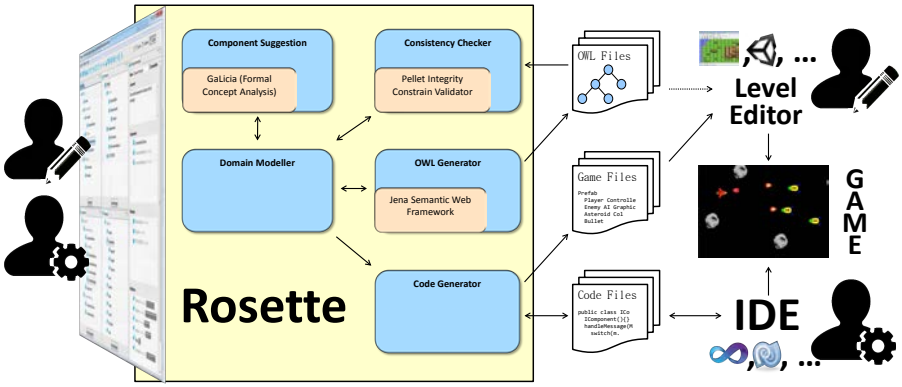


Fig. 1. General view of the *Rosette* architecture

structures such as finite state machines or behaviour trees are specified. The knowledge-rich information of the game domain allows us to detect inconsistencies, such as when an entity is tried to make an action that it is not supposed to do according to the game domain [6]. In this way, the game semantic model constitutes a central point of the process and a *verifiable game specification*.

3 Rosette

Although our methodology is valuable as-is, its main benefit comes when it is supported by tools that help in the construction of the game model, code generation, and consistence checking.

In that sense, we have developed a visual authoring tool called *Rosette* [2] that fills this gap. *Rosette* acts as a central point where designers and programmers come to an agreement (see figure 1). The tool is aware of these two different roles and presents different information and options depending on the role of the current user.

Though the current state of the application allows an iterative development, in the description that follows we will assume that we are using a waterfall model, just as we did in the previous section.

At the beginning of the development process, *Rosette* is used by designers. There they define a kind of conceptual hierarchy of the entities that comprise the game. *Rosette* allows adding to every entity a comment that acts as documentation and a set of other properties. In particular, designers may specify a set of attributes (*name*, *health* or *maxSpeed* to mention just a few) and a set of actions that the entity can perform (such as *shoot* or *move*). As the entities are organized hierarchically, the inheritance model applies and therefore the set of attributes and actions that an entity has available also contains the attributes and actions of all its predecessors in the hierarchy. As an analogy, we can say that the process is similar to defining a UML class diagram but without taking into account the peculiarities of programming languages but the game domain

itself, and therefore the final result is closer to an *ontology* than a class diagram (users at this point are designers not programmers).

The final result of this step may be seen as an specification of the game or, put it in other way, the contract between designers and programmers, because it specifies the set of entities that designers assure they will need to build the game levels and, therefore, the functionality programmers must develop.

The complete game domain (referring both designers' gameplay aspects and programmers implementation needs) is stored using OWL, a knowledge-rich representation that allows inferences and consistence checking. Entity verification allows to detect errors as soon as possible, instead of let them go down until the final source code and game execution, when it would be too expensive to solve them.

In the next step, programmers come to scene. Within *Rosette*, they change the view and available options to enrich the previous information with other knowledge that may be needed prior coding. This may even involve creating new entities that designers do not care about, such as cameras, waypoints or triggers, or populate the hierarchy with internal entities that will make the development phase easier.

After that, programmers may take a step further using *Rosette*: split entities into components. They, as experts, can manually specify the components that the game will have and then modelling the set of entities on the game domain according to them. The process is supervised by *Rosette*, and so, it checks whether the final result is complete, in the sense of assuring that all the attributes and actions an entity has are covered by the chosen components for that entity.

Moreover, *Rosette* may aid in the process of generating components [5, 4]. Instead of having developers creating the set of components from scratch, *Rosette* may give them a initial proposal to begin with. As the fully-fledged domain model allows us to use reasoning engines, it can be used together with Formal Concept Analysis (FCA) for extracting an initial set of components. FCA is a mathematical method for deriving a concept hierarchy from a collection of objects and their properties [1]. This may be useless in our case, because we do not have a collection of objects but a hand-made conceptual hierarchy and therefore it may be argued that FCA cannot help in the process. However this is not entirely true. In short, *Rosette* converts the game model (a conceptual hierarchy by itself) in a set of objects and feed them into FCA. The final result is not the original ontology but a (formal) concept hierarchy where each (formal) concept represents not one but maybe more than one original concept sharing a set of properties. Since a component is also a set of properties (actions and attributes) that may be used by several objects (entities), *Rosette* uses this (formal) concept hierarchy to automatically identify a component distribution that will fit the requirements of the provided entity ontology.

The candidate distribution is shown to the user through the visual interface, where he has the opportunity of modifying it by merging or splitting components, adding or removing extra functionality, changing names, etc.

At this point, the modelling phase is over and both designers and programmers start working independently on their respective tools: level editors and IDEs. In order to assure the integrity between the game domain created within *Rosette* and levels and code generated with external tools, *Rosette* has the capability of integration with two dif-

ferent target platforms: the well known Unity3D game engine and an in-house engine developed by the authors in C++ that is being extensively used in our master degree of videogames in the Complutense University of Madrid and supported by *Rosette* to use it in our classes [3]. So, once designers and programmers agree, the target platform is selected in *Rosette*, and it generates the input files for their respective tools. In case of Unity3D, the project is populated with a prefab per entity and a C# file for every component. Prefabs are preconfigured with the set of components specified in the domain model and the initial values of their attributes (if any). On the other hand, the source files act as skeletons with hooks where programmers will add the code that performs the specific actions assigned to the components. By contrast, if the selected target platform is our in-house game engine, a plethora of C++ files (both header and implementation files) is created and some XML files are generated containing the list components of every entity (*blueprints*) and the default values of their attributes (*archetypes*). *Rosette* has been designed to be extensible in the sense that many other target platforms may be added without too much effort.

Figure 1 summarizes the process: both designers and programmers use *Rosette* as a domain modeller tool. The game model is then used for consistency checking using OWL and for component suggestion using FCA. When the process is complete, *Rosette* code generator creates game files meant to be read by the used level editor and source files that will be enriched by programmers through IDEs.

The main advantages of *Rosette* for supporting the methodology are:

- *Join point between designers and programmers*: instead of having designers writing game design documents that are hardly read by programmers, *Rosette* acts as a union point between them because the model created by designers is the starting point for programmers work.
- *Easy authoring of the game domain*: designers are comfortable with hierarchical (not component-based) domains. Using *Rosette*, they are able to create the entity hierarchy enriched with state (attributes) and operations (actions).
- *Semantic knowledge stored in a formal domain model*: the game domain is transparently stored in a knowledge-rich representation using OWL.
- *Automatic component extraction*: using Formal Concept Analysis (FCA), *Rosette* may be used for suggesting the best component distribution for the entity hierarchy specified by designers and enriched by programmers [5, 4] and allows manually fine-tune that components.
- *Consistency checking of the game domain*: the semantic knowledge allows validation and automated consistency checking [9]. As an example, *Rosette* guarantees that manual changes done by programmers to the component distribution are coherent with the game domain [2].
- *Code sketches generation*: as in Model Driven Architectures (MDA), *Rosette* boosts the game development creating big code fragments of the components. This saves programmers of the boring and error-prone task of creating a lot of component scaffolding code. *Rosette* is platform independent in the sense that its code generator module is easily extended to incorporate other target platforms different than the two currently supported, Unity3D and plain C++.

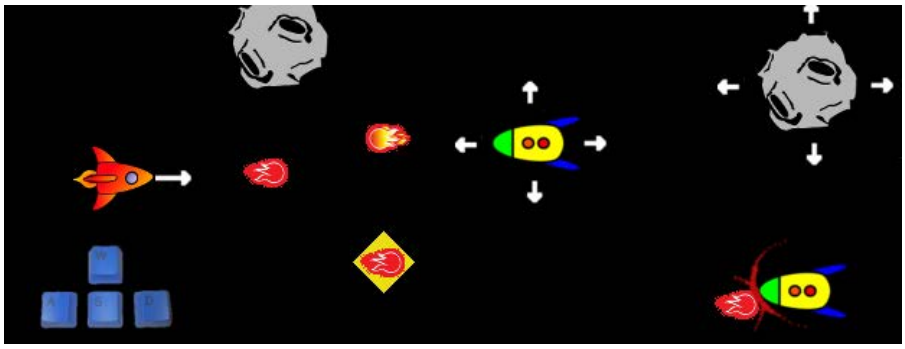


Fig. 2. Game to be developed

4 Development example

We have developed a small game (Figure 2) to show the application of the methodology and the use of *Rosette* with a practical example. The game we developed consists in a side scrolling shoot'em-up 2D arcade where the player controls a spacecraft that constantly advances throughout the level but whose movement can be altered by the player who also shoots bullets to enemies whilst avoiding collision of asteroids. Enemies of the game are also spacecrafts that move and shoot randomly appearing from the right side of the screen. Both enemies and the player lose health when collide with bullets or asteroids. Some of these asteroids are static whilst others move randomly appearing all of them from the right side of the screen. Finally, there are power ups in the level that increment ammunition of the player when they are caught.

Designers are responsible for defining the previous game more in depth, determining the details of the playability. According to our methodology, this means that they are in charge of creating a hierarchical distribution of entities in *Rosette*. Figure 3 shows a possible distribution. Entities are shown in the left-side of the picture. Entities shadow in grey (*GameObject*, *LevelElement*, *Bullet*, *PowerUp* and *Ship*) are important from the semantic point of view but they will surely not exist as such in the final game; their main purpose is to enhance the ontology knowledge, but only black entities (*Asteroid*, *PlayerBullet* or *Enemy*) will be instantiated in the game.

Once designers have decided the different entities of the game, they will also have to define the state and functionality of every entity in terms of attributes they have and actions they are able to carry out.

In our running example, designers identified six attributes (such as *position*, *health* or *numberOfBullets*) and three actions (*Move*, *Shoot* and *SufferDamage*). They are shown in the middle part of the figure 3.

The right side of this figure also shows the details of the *Enemy* entity, in terms of these attributes and actions. Using attributes, the designer specifies that any *Enemy* entity will have a state consisting of the number of hits that make the enemy die with *health* attribute, the concrete name of the entity with the *name* attribute, the number of bullets it has available through *numberOfBullets*, the *position* in the level and the

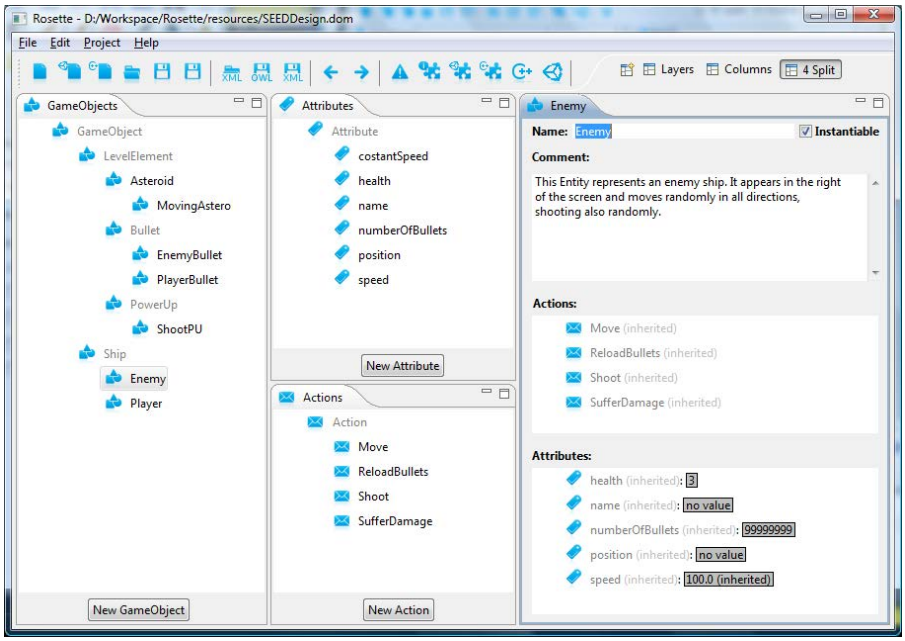


Fig. 3. Domain proposed by the designer for the game

speed of its movement. The designer can also set default values to the *archetypes* of the entities that will be assigned to all the *Enemy* entities when added in the level editor (for example, *3* for *health*). These values will be editable later in a per instance basis.

On the other hand, designers must also endow entities with functionality and, for this purpose, they should assign to entities the actions that they can carry out. Continuing with the example, *Enemy* entities can move through the level with *Move* actions, shoot with the *Shoot* action, reload bullets with the *ReloadBullets* action and lose health with *SufferDamage*.

Both *Enemy* and *Player* are very similar entities so their definitions would be almost the same, with small differences such as the *Player* is always moving forward through the level, what means it has to use the *constantSpeed* attribute that defines the vector that continuously modifies the position. In traditional developments it would not be easy to identify the similarities between both entities but with *Rosette* this fact can be seen at a glance due to the two of them inherit from *Ship*.

This domain created by the designers must be accepted by the programmers because in some way this will be a contract between programmers and designers. If programmers detect something critical from the implementation point of view they must communicate it to the designers before they accept the domain. In the case of this example, the domain proposed by the designers does not implies any problem for the implementation of the functionality so the programmers accept the given domain and are now responsible of adding all the extra information needed for the implementation of the

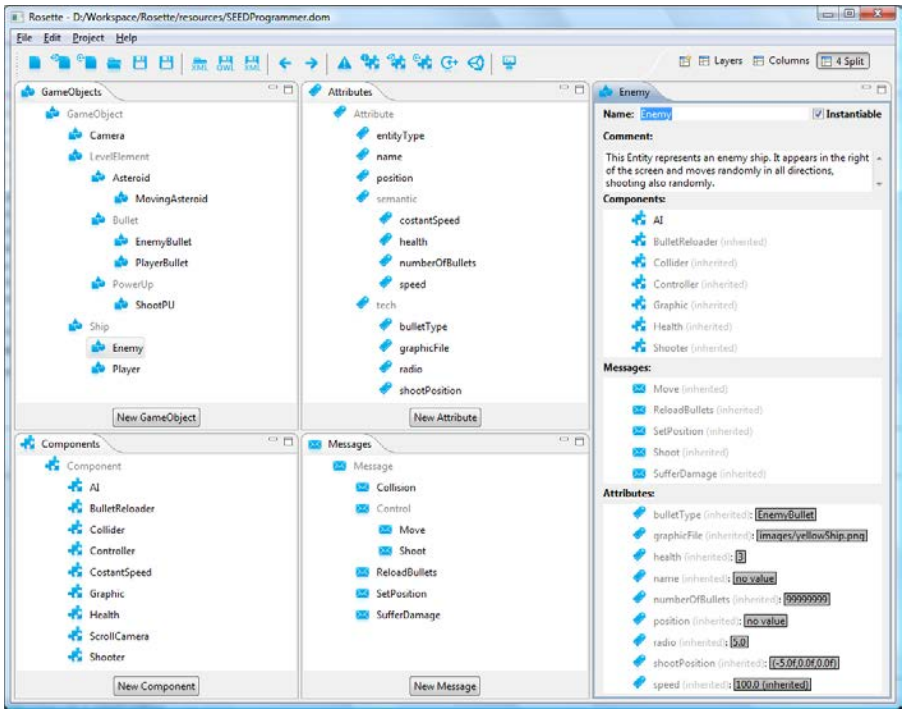


Fig. 4. Domain completed by the programmer for the game

game. This does not just suppose the addition of the software components but also the addition of new entities to represent abstract things, new attributes without semantic interest and possibly new actions that, from the programming point of view, correspond to messages that entities send and accept in the component-based architecture.

Figure 4 shows the next step of the game development where programmers have completed the game model given by the designers. In this figure it can be seen that programmers have added an extra entity to implement the game: *Camera*. It, obviously, represents the camera, which is responsible of deciding what must be rendered and will advance through the level keeping the player on focus.

Attributes have been also extended. A small hierarchy has been created, and new attributes have been added:

- *bulletType*: Kind of bullet that is shot. Different entities can shoot different bullets.
- *entityType*: Identify the type of the entity.
- *graphicFile*: Determines the resource to be rendered.
- *radio*: Let collision works with entities of different sizes.
- *shootPosition*: Relative position where bullets are shot. This way the *Shoot* functionality can be shared between spaceships with cannons in different positions.

Moreover, programmers added the new *SetPosition* action/message that modifies the position of an entity and the *Collision* message that is sent when two entities collide.

Component	Attributes	Actions	Belongs to
AI			Enemy, MovingAsteroid
BulletReloader	numberOfBullets		Enemy
Collider	radio	SetPosition	Asteroid, Enemy, EnemyBullet, Player MovingAsteroid, PlayerBullet, ShootPU
Controller	speed	Move	Enemy, MovingAsteroid, Player
ConstantSpeed	constantSpeed		Camera, EnemyBullet, Player, PlayerBullet
Graphic	graphicFile	SetPosition	Asteroid, Enemy, EnemyBullet, Player MovingAsteroid, PlayerBullet, ShootPU
Health	health	SufferDamage	Enemy, EnemyBullet, Player, PlayerBullet
Shooter	bulletType, numberOfBullets	ReloadBullets, Shoot	Enemy, Player
ScrollCamera			Camera

Table 1. Components created by programmers

From the semantic point of view probably these attributes and actions have no sense but they are totally needed to create reusable software.

Nevertheless, the more relevant task in *Rosette* for the programmers consists in distributing the state and functionality of the entities among components. With this aim, programmers may decide to make the component distribution by hand or they can use *Rosette* to obtain a candidate distribution for the current entities described in the domain and then they can adapt it as desired. Table 1 details the software components resulting from the final distribution of the state and functionality. An initial distribution was done using the component suggestion technique of *Rosette*, and then programmers applied some modifications that improved future reusability. Specifically, the technique could not detect some things because of the limited number of entities, but programmers can improve it using their experience. For example, in our game every renderable object also accepts collisions so *Rosette* identified both features as related and encapsulated them in the same component. However, the programmer, anticipating further steps of the development, split this component in *Graphic* and *Collider* components. This will allow to have, in later revisions of the game, decorative graphic entities (without collisions) or invisible entities (just collisions) that react to collision triggering some events or changing the game experience.

When programmers are pleased with the proposed distribution of the features, the next step is to take advantage of all the semantic knowledge introduced in *Rosette* to automatically generate all the scaffolding code in order to both designers and programmers can continue with their work and finish the game.

We have implemented this game in two platforms and programming languages. The first one was Unity [1] (and C#). The second one was created in C++. We have available a game skeleton that incorporates a minimal application framework that manages the main loop of the game, and a graphic server that encapsulates the inner workings of the

[1] http://unity3d.com/

Ogre render API[2]. It also had a small logic layer that manages component-based entities. The logic layer was able to load the description of the entities and the level map from external files. It, however, had no components, messages and entity descriptions, so for that reason we had to design the capabilities of every component from scratch as we have done previously. *Rosette* could generate the C++ code that is smoothly integrated within this logic layer. For the rest of the paper we will use the second implementation (C++) to explain the process as it is more complete and requires more effort.

Continuing with the role of the programmer, *Rosette* generates a big quantity of the code related with the management of the game logic that includes the entity management, the components and the message passing between components. Having these code assets, the task of the programmers in the game logic is reduced to implement the real functionality described in *Rosette* as actions/messages and some technical iterative functionality to communicate the logic with other modules such as the graphic, sound or physics modules. In this concrete example of game development, the 91,5% of the code related with the components and the message passing was autogenerated by *Rosette* what notably reduces the programming effort.

Figure 5 shows an example of what should be the work of the programmer in this step of the development for a simple action such as shooting. *Rosette* generates an empty function in the *Shooter* component and the code responsible of invoking it when the entity that contains an instance of this component receives a message of *Shoot* type. The programmer should then fill the body of this function where, if the entity has enough ammunition (*numberOfBullet*), a new parametrized entity is created (*bulletType*) in the corresponding position relative to the entity (*shootPosition*) and is removed after one second.

```
void Shooter::Shoot()
{
  if(numberOfBullet > 0)
  {
    CEntityFactory* ef = CEntityFactory::getSingletonPtr();
    CEntity* s = ef->instantiate(bulletType,
                                 getPosition()+shootPosition);
    ef->deferredDeleteEntity(s,1000);
    --numberOfBullet;
  }
}
```

Fig. 5. `Shoot` action.

Although the code assets generated for the programmers are a very important part of the game content created by *Rosette*, it also generates other content needed by the design tools, such as the level editor or AI editors. The generated content includes a

[2] http://www.ogre3d.org/

description of the entities in terms of attributes populated with default values that is used by the level editor to allow designers fill the level with instances of the described entities, allowing them to modify not just the position in the map but also the default values of the parameters, which were established in *Rosette*. This is a very simple game but if we wanted to improve it by adding more complicated behaviours, for example for final bosses, *Rosette* would create files describing the entities in function of attributes and the actions they are able to carry out. This way, AI tools facilitates the creation of behaviours for specific entities and also detects inconsistencies while designing. Summarizing, in this final step the designers will design the different levels of the game, making them enjoyable, whilst the programmers finish the implementation of the functionality requested in the previous step.

5 Related Work and Conclusions

Regarding related work in the use of explicit domain modeling for videogames we can mention the work by Tutenel *et al.* [7]. In this work they propose the use of an ontology about landscapes, buildings, and furniture that allows to procedurally generating content for a videogame. The ontology allows to integrate specialized generators (landscape, facade, furniture, etc.) to produced the different elements of a 3D environment, which are semantically coherent. A key difference with the work presented here is that we concentrate on the creation of behavior for the characters while they generate the static environment in a game.

The work presented in [8] complements the approach presented here, while we have detailed the process from the domain model to the source code, they concentrate on the use of the domain model to support designer tools. They show how the symbolic annotation of the environmental elements supports the creation of tools to incorporate information into the game design and development process. Such annotations can help to create information-rich interactive worlds, where designers can work with these environmental information elements to improve NPC (Non-Player Character) interactions both with the player and the environment, enhancing interaction, and leading to new possibilities such as meaningful in-game learning and character portability.

In conclusion, we have presented an initial version of a novel methodology for developing games in collaboration between programmers and designers, where a declarative model of the game domain is specified by designers and serve as the specification that programmers must implement. We have shown how this methodology can be supported by *Rosette*, both in authoring the domain model and elaborating it into actual code, for different platforms.

As future work we plan to fully develop the methodology as an iterative process where agile development is supported, facilitating the evolution of the domain model as new requirements are defined by designers, and helping programmers to iterate on the software elaborated from different versions of the domain model. We also plan to explore the tools and techniques that can exploit a declarative domain model from the designer point of view, for tasks such as consistency checking of the actual level maps in a game, scripting NPC behavior, debugging a level by procedurally generating solutions that can be automatically tested, and managing a library of reusable behaviors.

References

1. B. Ganter and R. Wille. Formal concept analysis. *Mathematical Foundations*, 1997.
2. D. Llansó, M. A. Gómez-Martín, P. P. Gómez-Martín, and P. A. González-Calero. Explicit domain modelling in video games. In *International Conference on the Foundations of Digital Games (FDG)*, Bordeaux, France, June 2011. ACM.
3. D. Llansó, M. A. Gómez-Martín, P. P. Gómez-Martín, P. A. González-Calero, and M. S. El-Nasr. Tool-supported iterative learning of component-based software architecture for games. In *International Conference on the Foundations of Digital Games (FDG)*, Chania, Greek, May 2013.
4. D. Llansó, P. P. Gómez-Martín, M. A. Gómez-Martín, and P. A. González-Calero. Iterative software design of computer games through FCA. In *Procs. of the 8th International Conference on Concept Lattices and their Applications (CLA)*, Nancy, France, October 2011. INRIA Nancy.
5. D. Llansó, P. P. Gómez-Martín, M. A. Gómez-Martín, and P. A. González-Calero. Knowledge guided development of videogames. In *Papers from the 2011 AIIDE Workshop on Artificial Intelligence in the Game Design Process (IDP)*, Palo Alto, California, USA, October 2011. AIII Press.
6. A. A. Sánchez-Ruiz, D. Llansó, M. A. Gómez-Martín, and P. A. González-Calero. Authoring behaviour for characters in games reusing abstracted plan traces. In Z. Ruttkay, M. Kipp, A. Nijholt, and H. Vilhjálmsson, editors, *Intelligent Virtual Agents*, volume 5773 of *Lecture Notes in Computer Science*, pages 56–62. Springer Berlin Heidelberg, 2009.
7. T. Tutenel, R. M. Smelik, R. Lopes, K. J. de Kraker, and R. Bidarra. Generating consistent buildings: A semantic approach for integrating procedural techniques. *IEEE Trans. Comput. Intellig. and AI in Games*, 3(3):274–288, 2011.
8. M. Youngblood, F. W. Heckel, D. H. Hale, and P. N. Dixit. Embedding information into game worlds to improve interactive intelligence. In P. A. González-Calero and M. A. Gómez-Martín, editors, *Artificial Intelligence for Computer Games*. Springer, 2011.
9. X. Zhu and Z. Jin. Ontology-based inconsistency management of software requirements specifications. In P. Vojtáš, M. Bieliková, B. Charron-Bost, and O. Sýkora, editors, *SOFSEM 2005*, LNCS 3381, pages 340–349. Springer, 2005.

Stories from Games: Content and Focalization Selection in Narrative Composition

Pablo Gervás

Universidad Complutense de Madrid,Spain,
pgervas@sip.ucm.es,
WWW home page: http://nil.fdi.ucm.es

Abstract. Game logs could be exploited as a source for reflection on user performance, with a view to improvement or simply the additional entertainment of reminescing. Yet in their raw form they are difficult to interpret, and sometimes only specific parts of the game are worth reviewing. The ability to produce textual narratives that rework these logs (or the interesting parts of them) as stories could open up this wealth of data for further use. This paper presents a model of the task of narrative composition as a set of operations that need to be carried out to obtain a linear sequence of event descriptions from a set of events that inspire the narration. As an indicative case study, an initial implementation of the model is applied to a chess game understood as a formalised set of events susceptible of story-like interpretations. Operating on simple representations of a chess game in algebraic notation, exploratory solutions for the tasks of content selection are explored based on a fitness function that aims to reflect some of the qualities that humans may value on a discourse representation of a story.

1 Introduction

The aftermath of a well-played game usually invites a process of reflection whereupon certain parts of the game are replayed in the mind (or on a blackboard or even on video). The purpose is sometimes to understand better a particular event in the game and sometimes simply to rejoice in it. This process can lead to improvements in performance, or it can be useful as a teaching aid. On a similar vein, digital entertainment generates a wealth of information in terms of traces of user activity in the form of game logs. These logs could be exploited as a source for reflection on user performance, with a view to improvement or simply the additional entertainment of reminescing. Yet in their raw form these logs are difficult to interpret, and reliving them in their original form would be too time consuming and not homogenously fruitful, as sometimes only specific parts of the game are worth replaying. The ability to produce textual narratives that rework these logs (or the interesting parts of them) as stories could open up this wealth of data for further use.

Automated composition of narratives from data is budding line of research within computational narratology, but elementary techniques are already available that can provide an initial approximation to this task [1–5]. The challenge

to undertake is to provide a computational model of the traditional concept of *raconteur*: someone who can identify something that actually happened, leave out the boring bits, focus on the interesting ones, and tell it in a way that makes it entertaining. The task itself can be broken down into several stages:

1. he selects the fragment of real life he wants to tell
2. he filters out the non-relevant parts to obtain his initial material
3. he imposes a certain order on the events that he wants to tell (from here on he has a narration sequence)
4. he decides where in that sequence to insert the descriptions he intends to use from his initial material (he now has a sequence that switches between narration and description)
5. he decides how much of that sequence he can assume that his listeners will infer without effort from the rest
6. he produces the right form for the remaining sequence

Steps 3 and 4 may be somewhat mixed or carried out in reverse order, depending on the style of the raconteur. Step 5 may be ommited. Step 6 involves linguistic skills that are a huge problem in themselves.

The present paper attempts to address these challenges from an engineering point of view: given an exhaustive record of all moves made in a given game, find a way of telling what happened as a linear sequence that may be later converted into text, trying to maximize coverage, minimize redundancy, and achieve a certain natural fluency. Chess has been chosen as an initial case study because it provides a finite set of characters (pieces), a schematical representation of space (the board) and time (progressive turns), and a very restricted set of possible actions. Yet it also allows very elementary interpretations of game situations in terms of human concepts such as danger, conflict, death, survival, victory or defeat, which can be seen as interesting building blocks for story construction.

The paper reviews existing models related to the narrative composition task, describes the proposed computational model, presents the case study for narration of chess games and finishes with discussion and conclusions.

2 Previous Work

A number of models of related tasks and elements arising from different fields of research are reviewed in this section to provide background material for the discussion. Due to the breadth of fields under consideration, exhaustive review in any one of them is beyond the scope of the paper. An attempt has been made in each case to gather here the set of elementary concepts in each field that are relevant for the understanding of the arguments in the paper.

2.1 Narratology

According to many theorists, narrative has two components: what is told (what narrative is: its content, consisting of events, actions, time and location), and the

way it is told (how the narrative is told: arrangement, emphasis / de-emphasis, magnification / diminution, of any of the elements of the content). These have been named different ways by different researchers, story and discourse, *histoire* and *discours*, *fabula* and *sujzet*. There are alternative analyses that postulate different subdivions. Even between theories that agree on having just two levels of analysis there seem to be many subtleties that cast doubt on whether the same thing is meant by the different words. This presents a serious obstacle for researchers from the computational field trying to address the treatment of stories in any form. In order to avoid ambiguity, we will restrict our analysis here to three levels of conceptual representation of a story, and refer to these as the *story* (the complete set of what could be told, organised in chronological order of occurrence), the *plot* (what has been chosen to tell, organised in the order in which it is to be told) and the *narrative* (the actual way of telling it).

Narratologists, who specialize in the study of narrative, consider the concept of *focalization* [6] as the way in which a narrator restricts what he is telling about a particular scene to what might have been perceived by someone present in that scene. This may be one of the characters if the scene is told in the first person, or the narrator himself as if he had been present (if the story is told in the third person). This has an interesting implication in the fact that, through focalization, narrative discourse (and thereby the structure of stories) is influenced by the perception of space: events that take place simultaneously in different locations that cannot be perceived at the same time (this may be different cities but also different neighbouring rooms separated by a wall) usually require different narrative threads.

2.2 Natural Language Generation

The general process of text generation takes place in several stages, during which the conceptual input is progressively refined by adding information that will shape the final text [7]. During the initial stages the concepts and messages that will appear in the final content are decided (*content determination*) and these messages are organised into a specific order and structure (*discourse planning*), and particular ways of describing each concept where it appears in the discourse plan are selected (*referring expression generation*). This results in a version of the discourse plan where the contents, the structure of the discourse, and the level of detail of each concept are already fixed. The *lexicalization* stage that follows decides which specific words and phrases should be chosen to express the domain concepts and relations which appear in the messages. A final stage of *surface realization* assembles all the relevant pieces into linguistically and typographically correct text. These tasks can be grouped into three separate sets: *content planning*, involving the first two, *sentence planning*, involving the second two, and surface realization. An additional task of *aggregation* is considered, that involves merging structurally or conceptually related information into more compact representations ("Tom fled. Bill fled." to "Tom and Bill fled." or "the boy and the girl" to "the children"). Aggregation may take place at different levels of the pipeline depending on its nature.

3 A Computational Model

The type of narrative that we want to address in this paper involves a linear sequential discourse where only a single event can be told at any given point. Yet reality is not like that. Events to be reported may have happened simultaneously in physically separated locations, and constitute more of a cloud than a linear sequence, a volume characterised by 4 dimensional space time coordinates. Composing a narrative for such an input involves drawing a number of linear pathways through that volume, and then combining these linear pathways (or a selection thereof) together into a single linear discourse. This type of linear pathway is sometimes referred to as a *narrative thread*. From a narratological point of view, this task can be related to the identification of appropriate focalization decisions for conveying a given material. Focalization, understood as the decision of which character the narration should follow, and how much of the environment around him at each point should be conveyed to the reader of the narrative, divides the perception of reality into individual fibres (one for each possible focalizer character) that are linear and sequential in nature. For each character involved in the set of events to be conveyed, a possible focalization fibre can be drawn.

To provide a preliminary benchmark for the various intuitions outlined in the rest of the paper the simplest approximation to a case study that could be conceived is described in this section. Characters will be chess pieces. To model the way humans tell stories in terms of narrative threads, based on subsets of the events that can be said to be experienced by a particular character, each piece is assigned a field of perception corresponding to a N x N square around the position of the board in which it is standing. The value of N is considered the *range of perception* of that particular piece. For a given piece with range of perception N, all possible partitions of the chess board in N x N squares would be possible locations in our world model - depending on the positions of characters on the board. The set of events that we would be considering is the set of all moves involved in a given game.

A basic software implementation has been written that reads a description of a chess game in algebraic notation (see Table 1) and builds for it a representation in terms of threads focalized on a given character assuming a certain range of perception around him.

Events are triggered by pieces moves. Whenever a piece moves, this constitutes an event for the piece itself, for any other piece captured during the move, and for any other piece that sees either the full move, the start of the move or the conclusion of the move.

Fibres for each of the pieces are built by collecting event descriptions for those moves that they are involved in or they see. The same event may get described differently in different fibres depending on the extent to which the corresponding focalizer is involved in it.

1. e4 c5	16. Bxe2 Be6
2. Nf3 d6	17. Rfd1 Rfd8
3. d4 cxd4	18. Bc5 Rd5
4. Nxd4 Nf6	19. b4 a5
5. Nc3 g6	20. Bf3 Rxd1+
6. Be2 Bg7	21. Rxd1 e4
7. Be3 O-O	22. Bxe4 Bxc3
8. O-O Nc6	23. Bxc6 Rc8
9. h3 d5	24. b5 Bxa2
10. exd5 Nxd5	25. Bd4 Bb4
11. Nxd5 Qxd5	26. Be5 Be6
12. Bf3 Qc4	27. b6 Rxc6
13. Nxc6 bxc6	28. b7 Rb6
14. c3 e5	29. Rd8+
15. Qe2 Qxe2	1-0

Table 1. Algebraic notation for an example chess game

3.1 Representing the Data

An *event* is something that happens with a potential for being relevant to a story. Events occur at a specific location at a given moment of time. They may have preconditions and postconditions. In order to have a generic representation, each event is considered to have an associated *event description* that allows both a descriptive and a narrative component. The descriptive component of an event description contains predicates that describe relevant preconditions. The narrative component of an event describes the action of the event itself, but it may also contain additional narrative predicates describing the effect of the action. For instance, if someone moves from one place to another, elements at the original location may be left behind, and elements in the final location appear. These are not considered separate events but included in the description of the moving event.

A *fibre* is a sequence of events that either involve or are seen by a given character. It represents a focalized perception of the world. The extent of information that is included in the events for a given fibre is defined by the range of perception that is being considered and by the presence of any obstacles to perception in the surrounding environment. It may also be affected by the direction in which the focalizer is facing, or where he is focusing his attention. As these further refinements require advanced authorial decisions, we decide that the standard representation should include all the information within perceptive range, and leave the decision of whether to mention it to a later stage of composition.

The task of *heckling*[1] involves establishing the range of perception, tracking the set of all possible characters involved in the events to be narrated, and

[1] From the process of extracting a set of fibres from flax or fleece, to be later spun into yarns.

for each character constructing a fibre representation that includes descriptions of all the events that the character initiates, suffers or perceives. These event descriptions will appear in the fibre in chronological order.

3.2 A Computational Procedure

From a given game, a large number of fibres, longer or shorter, and at different ranges of perception can be produced. Most of them overlap one another, and most of them will contain descriptions of events that are not interesting in themselves and not relevant to the overall flow of the game. To select among them a selection of subsets that might constitute acceptable tellings of the game we have considered an evolutionary approach. This allows us to concentrate our effort on the definition of fitness functions that capture desireable qualities of a good story, both at the domain level and in terms of the structural properties it should have. We therefore need to define an initial population of fibres, and a set of operators for crossing and mutating them, using these fitness functions to rate successive generations. This would constitute an evolutionary approach [8] to the task of content and focalization selection.

The concept of a *draft* that holds the current best solution for a given telling of the game under consideration and which gets progressively modified towards an optimal solution, is fundamental to the proposed model. The concept of *reviser*, a module that operates on a draft to progressively improve it, captures this concept of progressive modification. The model will operate not on a single draft but over a *population* of candidate drafts.

We allow a set of alternatives for the creation of the drafts in the initial population. To this end we introduce the concept of *babbler*, a module in charge of producing an initial draft. By allowing a population of babblers to produce the initial population, we introduce the possibility of relying on more than one technology to produce them. Solutions for babblers that have been explored so far generate a set of single fibre drafts which are then combined into more complex fibres by subsequent stages of the system.

Drafts need to be evaluated for conformance with the desired qualities for the telling of the game, and the results of this evaluation need to be taken into account in any subsequent operations on the draft. The concept of a *judge*, a module capable of evaluating partial results according to desired criteria, captures this idea. The various judges assign scores on specific parameters (designed to isolate structural properties such as redundancy, coverage, cohesion or overall interest):

- *uniqueness*, measures the numbers of events made or seen by the focalizers of the fibres involved in a telling that are not already covered by some other fibre in the telling, as a percentage of the total number of narrative predicates in the yarn (which should correspond to all the events mentioned),
- *density* measures the ratio between captures and narrative predicates (the number of narrative predicates that are captures as opposed to just moves),
- *coverage*, measuring the percentage of moves of the game that are actually covered by a given telling,

– *cohesion*, measuring the ratio between the number of moves covered by the longest fibre in a telling and the number of moves covered by the telling (not necessarily the game, as parts of it may have been lost, and that is measured elsewhere by the coverage metric),
– *number of focalizers*, assigning high scores to tellings that have a number of different focalizer within a specified range (currently set between 2 and 6).

The definition of density is based on the assumption that captures constitute more emotionally charged events than other moves. An overall score for each draft is obtained by averaging the value of all individual scores received by the draft. Weighted combinations may be introduced at later stages if they are seen to improve the quality of the final results.

Evolutionary approaches rely on a set of cross over and mutation operators [8]. In our framework, cross over functionality is introduced by *mixers* and mutation functionality by *revisers*.

Two types of mixers are considered:

– one that given two different tellings, generates another two in which fibres ocurring for the same focalizer in both, but with different coverage of the actual set of events it took part in (either as an active participant or as an observer),[2] have been swapped
– one that given two different tellings produces a single telling that combines all the significantly different[3] fibres found in both

During one generation, each mixer is applied as many times as there are drafts in the population, and each time it combines two drafts selected at random.

Revisers rely on scores assigned by judges to introduce changes to drafts. Reviser act as mutator operators, taking an existing draft and producing a new draft that differs from the original in some way. The set of revisers includes:

– dropping certain fibres from a telling (for each available draft, produce a new draft by dropping fibres at random with a likelihood of 10 %)
– trimming certain fibres from a telling (for each available draft, produce a new draft with the same number of fibres but dropping event descriptions from the original fibre at random with a likelihood of 5 %)

During one generation, each reviser is applied to all the drafts in the population.

At the end of each generation, the size of the population has multiplied significantly. The resulting set of drafts are scored, and only top scoring ones are allowed onto the next generation. The number of drafts allowed to survive from each generation is a configuration parameter currently determined empirically.

[2] Differences in coverage may arise from the fibre having been trimmed as a result of revision or by virtue of having been originally constructed based on a different value for the range of perception.

[3] Two fibres are considered significantly different unless the set of events covered by one subsumes the set of events covered by the other.

3.3 Results

Running the system produces a population of drafts that constitute possible alternative ways of telling the story by focusing on one or another set of characters. With a view to explainig the intuitions behind the described evolutionary set up, an example of system run with a simple configuration is described below. These tellings are produced by a combination of a babbler that produces a different draft for each available fibre, a mixer that combines drafts by merging all their constituent fibres, and the full set of judges described.

For instance, for the game shown in Table 1, a number of possible tellings are summarised in Table 2. The top scoring single fibre candidate is draft 6, telling the story of the game that focuses on the second white pawn using a range of perception of 3. This is the top scoring draft on coverage (78) that has a single fibre. This pawn sees much of what happens in the game. It also has a top score on uniqueness (100), because no one else is reporting the same things. The overall top performer is draft 114, that tells the game through the eyes of the fifth black pawn and the left white bishop. The pawn uses a smaller range of perception (2) than the bishop (3). They achieve a top score on coverage of 96, but get penalised on uniqueness (73), because there is a large number of events that are reported by both of them (in bold in their list of moves). For completeness, data for two additional drafts (numbers 97 and 124) are shown, both with an overall score of 70 but using two and three focalizers respectively. Maximum coverage (98) is achieved with three focalizers in 124, but uniqueness drops down to 53, bringing the overall score down. This example shows how the metrics of the different judges interact to achieve a balance that leads to successful tellings of the game.

4 Discussion

The proposed system constitutes a computational implementation of two tasks relevant to narrative compostion: selection of relevant material by filtering out less relevant information (corresponding to the content determination subtask in natural language generation pipelines) and determination of appropriate focalization choices (a subject studied by narratology, though usually considered only during analysis of literary texts in terms of how focalization has been established by human authors). From the point of view of natural language generation, the task of content determination for narrative had not to our knowledge been explored computationally before. From the point of view of narratological theory, the model as described captures the concept of focalization as a computational generative task, and relates it closely to the input data, the decision criteria, and the computational processes being modelled.

4.1 Comparison with Other Narrative Composition Systems

A number of related efforts exist to automatically derive narratives from sport games [2],[3],[4]. These efforts operate on input data in the form of statistics on

Dn	Fc	Un	Cv	Ch	Dn	Os	# F	# m	Piece	Pr	List of moves
6	50	100	78	100	14	68	1	45	wp2	3	1, 2, 5, 6, 7, 9, 11, 13, 15, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57
114	100	73	96	89	10	73	2	55	bp5	2	2, 4, **6**, 8, 10, 12, 14, **16**, **18**, **19**, **20**, **21**, **22**, **24**, **25**, **26**, **28**, **30**, **31**, **32**, **35**, **36**, **39**, **40**, **42**, **43**
									lwb	3	1, 3, 5, **6**, 7, 9, 11, 13, 15, **16**, 17, **18**, **19**, **20**, **21**, **22**, 23, **24**, **25**, **26**, 27, **28**, 29, **30**, **31**, **32**, 33, **35**, **36**, 37, 38, **39**, **40**, **42**, **43**, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57
97	100	73	92	73	14	70	2	53	wp7	3	1, 3, 5, 6, 7, 11, 13, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 28, 29, 30, 31, 33, 35, 36, 39, 40, 41, 42, 43, 45, 49, 51, 57
									lbb	3	2, 4, 6, 8, 12, 14, 16, 18, 19, 20, 21, 22, 24, 25, 26, 28, 32, 34, 35, 36, 37, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57
124	100	53	98	87	13	70	3	56	wp7	3	1, 3, 5, 6, 7, 11, 13, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 28, 29, 30, 31, 33, 35, 36, 39, 40, 41, 42, 43, 45, 49, 51, 57
									bp6	2	4, 8, 10, 12, 14, 18, 19, 20, 21, 22, 24, 28, 32, 34, 36, 40, 42, 44, 48, 51, 52, 57
									lwb	3	1, 3, 5, 6, 7, 9, 11, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57

Table 2. Examples of tellings: draft number (Dn), number of fibres (# F), number of moves (# m), range of perception of the fibre (Pr), focalizer name (Piece), and scores for focalizer count (Fc), uniqueness (Un), coverage (Cv), cohesion (Ch), density (Dn) and overall score (Os).

a given game, and produce texts in the manner of newspaper articles covering similar games. These efforts, for instance, are heavily mediated by the set of data they start from, which arises from purpose specific abstraction and filtering of the set of real world events, driven by the purpose of the desired story. Additionally, the choice of news article as target form imposes stylistic choices in terms of focalization (the stories are not focalized on specific characters) and the rhetorical structure (for instance the need to provide a summary of the game at the beginning, then review the highlights, then comment on any relevant details). The input data considered in these cases already involve a severe operation of abstraction, in the sense that they do not constitute raw perceptions of reality but purpose-specific abstractions such as statistics on percentage of succesful attempts, relative number of fouls between opposing teams, or total time that a team had control of the ball. In such cases, a large portion of the effort of converting the real life data into story material has already been achieved by the stage of drawing statistics from the game. As such, they differ greatly from the type of input data and output text being considered in this paper. For the model described here, input data would be a complete log of all events that took place in the game, and the output text would be closer to a multi-character novel describing the game, or the script of a film telling the story of the game.

The task of narrative composition has also been explicitly addressed by Hassan et al [1] and Gervás [5]. Hassan et al addressed the task of generating stories from the logs of a social simulation system. Gervás focused on the task of generating a natural language rendering of a story extracted from a chess game for a given set of pieces and with focalization set at a prefixed range of perception (squares of size 3 were used). An analysis of the differences between these systems is best carried out in terms of the 6 stages described in section 1 for the task of a *raconteur*.

Of those stages, step 1 would correspond to selecting a particular game, or particular social simulation, and in all cases is already decided in the input received by the systems. The system described in the present paper models step 2, corresponding to filtering out the non-relevant parts. This step was not addressed by Hassan et al [1]. The procedure employed in the present case relies heavily on the fitness functions implemented in the judges to establish the filtering criteria, though it is the mixers and the revisers that carry out the executive work of tailoring the drafts. Step 3 (imposing a particular order on events to be told) is addressed by all three systems, with Gervás [5] being the one that devotes most effort to it. The present system relies on the solution for step 3 developed in Gervás [5]. This step presents interesting features in the context of the case study on chess, and is addressed below under 4.2. Step 4 (generating and inserting descriptions at appropriate places in the narrative sequence) has so far only been addressed in Gervás [5]. Step 5 (establishing how much of the intended content can be omitted and left for the reader to infer) requires a rich model of what the reader knows and has so far been sidestepped by all previous research efforts. Step 6 (rendering the resulting conceptual discourse as natural language) was addressed by Hassan et al [1] in terms of template-based NLG and

by Gervás [5] with a more refined NLG system that included referring expression generation, lexical and syntactic choice, and grammar-based surface realization.

4.2 Generalizing beyond Chess

Chess games present the advantage of having very clear temporal and spatial constraints, and constituting at heart a sketchy representation of one of the most dramatic settings for human experience: war. In that sense, it provides a good ground for simple experiments, and it is not intended as a contribution in itself but as an illustrative example of the operation of the model of sufficient simplicity to be describable within the size restrictions of a paper such as this. Two aspects are identified as problematic with the use of chess games as a case study, one related to the tasks of content selection and focalization choice and one more related to the task of establishing a sequential order on the narrative (corresponding to step 3 mentioned in section 1).

First, adequate representation of opportunities and threats in a chess game involves some serious representational challenges [9]. For more elaborate experiments, the fitness functions employed would have to be refined to better capture this complexity. Although significant progress may be made on this point, the effort invested is unlikely to lead to compelling narratives or narratives that bring insight on the task of narrative composition. It might be more fruitful to identify richer domains for further case studies, and to apply the engineering effort to whatever elements are considered of narrative interest in them.

Second, the chronological structure of a chess game is in truth purely sequential, in contrast with the sets of events that would be considered when narrating from real life events. This has not been considered a serious obstacle in as much as focalization breaks up the set of events into separate views corresponding to different fibres, and the numbering or moves in the game provides a good indication of relative chronology of events within any given fibre and across fibres. For these reasons, it is considered advisable to explore further investigation of the suitability of the model when applied to case studies in other domains that are richer in terms of their representation of time and space and that may lead to more compelling narratives with a stronger human interest. The search for alternative domains is made difficult by the need to obtain for them reliable records of all events, both relevant and irrelevant to the story that may be told. Only if this condition is satisfied can the corresponding problem be considered equivalent to the human task that we want to model.

These two points suggest strongly that logs of video games would be a very good domain of application, as they satisfy all the described requirements. Situations and events of interest can be established for each particular game, and means would have to be found for identifying them computationally. Existing research on plan recognition [10–12] may be of use in addressing this task. Logging procedures can be defined for the chosen game, to ensure that game events, both relevant and irrelevant to the story that may be told, get registered.

5 Conclusions

The evolutionary solution presented in this paper constitutes a valuable contribution to the growing corpus of research on narrative composition. The tasks of content determination and focalization choice in narrative had not been addressed computationally before. The evolutionary approach provides a breakdown into subtasks that has lead to interesting insights in terms of specific knowledge-based operations that need to be carried out during composition. These operations provide a first computational approximation to decision making about focalization, as opposed to analysis of how focalization has been configured. These tasks can also be confortably placed within generally accepted task divisions for natural language generation. The preliminary implementation for the particular case of a chess game has shown that a set of possible sequential tellings with acceptable structural properties can be obtained from a formal log of considerable complexity. This opens the possibilities for addressing richer case studies for more complex games.

References

1. Hassan, S., León, C., Gervás, P., Hervás, R.: A computer model that generates biography-like narratives. In: International Joint Workshop on Computational Creativity. London. (2007)
2. Allen, N.D., Templon, J.R., McNally, P., Birnbaum, L., K.Hammond: Statsmonkey: A data-driven sports narrative writer. In: Computational Models of Narrative: AAAI Fall Symposium 2010. (2010)
3. Lareau, F., Dras, M., Dale, R.: Detecting interesting event sequences for sports reporting. In: Proc. ENLG 2011. (2011) 200–205
4. Bouayad-Agha, N., G.Casamayor, Wanner, L.: Content selection from an ontology-based knowledge base for the generation of football summaries. In: Proc. ENLG 2011. (2011) 72–81
5. Gervás, P.: From the fleece of fact to narrative yarns: a computational model of narrative composition. In: Proc. Workshop on Computational Models of Narrative 2012. (2012)
6. Genette, G.: Narrative discourse : an essay in method. Cornell University Press (1980)
7. Reiter, E., Dale, R.: Building Natural Language Generation Systems. Cambridge University Press (2000)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1989)
9. Collins, G., Birnbaum, L., Krulwich, B., Freed, M.: Plan debugging in an intentional system. In: Proc. IJCAI 1991. (1991) 353–359
10. Fagan, M., Cunningham, P.: Case-based plan recognition in computer games. In: Proc. of the Fifth ICCBR, Springer (2003) 161–170
11. Ramirez, M., Geffner, H.: Plan recognition as planning. In: Proc. of IJCAI-09. (2009)
12. Synnaeve, G., Bessière, P.: A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft. In: Proc. of AIIDE 2011, Springer (2011) 79–84

Playability as Measurement of the Interaction Experience on Entertainment Systems

J. L. González Sánchez, F. L. Gutiérrez Vela
GEDES – Universidad de Granada
joseluisgs@ugr.es and fgutierr@ugr.es

Abstract. Nowadays, video games are the most economically profitable entertainment industry. The nature of their design means that user experience factors make design and / or evaluation difficult using traditional methods commonly used in interactive systems. It is therefore necessary to know how to apply Playability in order to design, analyse, optimize and adapt it to a player's preferences. In this work we present a way to perform UX based on Playability techniques evaluate the entertainment and the game experience on video games. The aim is to easily and cost-effectively analyse the different degrees of Playability within a game and determine how player experience is affected by different game elements.

Keywords: Computer Science, Video Games, Entertainment Systems, User Experience, Playability, Hedonic Factors, Emotions.

1 Introduction

Video games are highly interactive systems whose main goal is to entertain users (players) that interact with them in order to have fun. Nowadays, video games are the most economically profitable entertainment industry.

User Experience (UX) is understood as a set of sensations, feelings or emotional responses that occur when users interact with the system. UX focuses more on the subjective aspect of the interaction process, and goes beyond the traditional study of skills and cognitive processes of users and their rational behaviour when interacting with computers [1, 2]. Due to the nature and design of videogames, user experience should be enriched by recreational, emotional, cultural, and other subjective factors that make analysis and evaluation difficult using traditional methods commonly used in interactive systems. Player eXperience (PX or User Experience in Video Games) depends not only on the product itself, but also on the user and the situation in which he or she uses the product. Players cannot be designed. Players are different. User Experience (UX) should be taken into account throughout product development (hardware or software), so as to achieve the optimum experience for the player. The current importance of video games and entertainment systems in society justifies the need for models that characterize the overall experience, and mechanisms for designing and analysing the Experience throughout the video game development process become a must.

The purpose of this work is to present Playability as measurement of the interactive experience with video games and entertainment systems, and uses it for the evaluation of the enjoyment and entertainment on interaction systems. This work reflects the importance of a Playability to evaluate the final experience for developing more

efficient and successful products in terms of entertainment and amusement, a crucial factor to improve the final satisfaction of users. In first second point, we discuss about the User Experience in Video Game, especially how emotions and cross-cultural factors can affect to the overall experience in games. Then, we present a Playability to characterize the User Experience in video games. Later, we apply the Playability Model in a Video Game Evaluation to analyse the experience. Finally, we discuss about our proposals in conclusion and future works related with this chapter will be presented.

2 User Experience in Video Games and Entertainment Systems

Analysing the quality of the experience on video game or entertainment systems purely in terms of its Usability is not sufficient; it is important to consider not only functional values but also a set of specific non-functional values, given the hedonic properties of video games: emotional response, social and cultural background, etc. User Experience in video games is enriched by the playful nature of game systems, together with the characteristics that give identity to the game, such as game rules, goals, challenges, rewards, GUI, dialog systems, etc., which are unique to each game and make the experience with every game different for each player, who will also have different experiences from one another. In the field of video games, the main goal is much more indefinite and subjective than in other interactive systems such us Desktop Systems: to feel good and have fun, in other words: 'entertainment'.

The property that characterizes the Player Experience or User Experience in entertainment systems is commonly called *Playability* and it can be used in the design process or at the evaluation stage to ascertain the experience that players feel when playing a video game. *Playability* is a term used in the design and analysis of video games that describes the quality of a video game in terms of its rules, mechanics, goals, and design. It refers to all the experiences that a player may feel when interacting with a game system. Playability is a key factor in the success of the video game due to its ability to ensure that the game engages, motivates and entertains the player during playing time. The most important references on playability are compiled in [3, 4, 5, 6]. Playability is a live topic in the scientific community; it has been studied from different points of view and with different objectives. In this section, we briefly describe the most representative works for each specific line of research.

It is worth mentioning the work of Rollings [7] which contains three key elements for identifying the playability of a videogame. These are: Core Mechanics, Storytelling & Narrative and Interactivity: set of elements that the player can see, hear and interact with in the virtual world. For Swink [8], playability is based on the combination and proper structuring of the game elements during the play-time. Furthermore, Akihiro Saito [9] indicates that the player experience is identified by "Gamenics": the quality of play, the quality of the platform on which a game runs and the mechanics of the game (GAme + MEchanics + electroNICS). Lazzaro [10] propose that one of the secrets of playability is the management of emotions and the

motivation of the player. Regarding "Flow" Theories, in [11] connected "a collection of criteria with which to evaluate a product's gameplay of interaction".

Other works on Playability and player experience define playability as "the usability in the context of video games", in which usability is understood as a traditional property of the UX. One of the most referenced works in this area is Federoff's proposal [12]. Following the line of heuristics and evaluation criteria of playability and usability in video games are the works by [13, 14]. Some interesting works focus more on how to evaluate the player experience applying biometric techniques [15] and gameplay metrics [16].

Björk's proposal offers a set of patterns and guidelines to game developers to increase the usability (playability) in video games [17]. But, it is important to readapt the experience to the user cross-cultural and location issues [18]; or promotes the social game implementing better mechanics for the collaboration among players [19].

Another important research line is the one which uses questionnaires to assess the user experience. The most significant is Game Experience Questionnaire (GEQ) [20, 21]. Moreover, it is possible to find style guides that promote the playability and accessibility in video games [22, 23]. Playability is a crucial factor, because to have the opportunity of combine accessibility techniques to properties to characterize and improve the entertainment of the player with the video game.

3 Playability Model to Characterize the Experience

As previously mentioned, the User Experience is characterized by two main points of view: process of use and product quality development. These are enhanced and enriched by the user's emotional reactions, and the perception of non-instrumental qualities. Playability is based on Usability in video games, but in the context of video games it goes much further. Furthermore, Playability is not limited to the degree of 'fun' or 'entertainment' experienced when playing a game. Although these are primary objectives, they are very subjective concepts. It entails extending and formally completing the characteristics of User Experience with players' dimensions using a broad set of attributes and properties in order to measure the Player Experience, see Fig. 1. There is a clear need for a common or unambiguous definition of playability, attributes to help characterize the player experience, properties to measure the development process of the video game, and mechanisms to associate the impact/influence of each video game element in the player experience. We consider this a significant lack, since the different definitions of playability require different criteria to measure it: there are no universals.

In this work we present a playability model to characterize PX in video games, introducing the notion of facets of playability, which integrates methodological approaches into the game development process. To achieve the best player experience, they propose a set of attributes to characterize the playability. Playability is defined as *'a set of properties that describe the Player Experience using a specific game system whose main objective is to provide enjoyment and entertainment, by being credible and satisfying, when the player plays alone or in company'* [6].

Playability describes the Player Experience using the following attributes and properties:

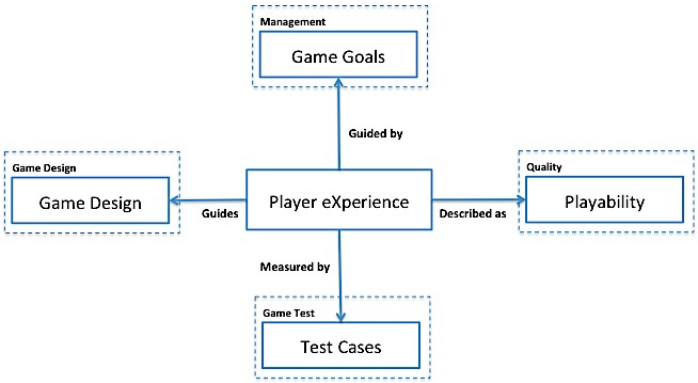


Fig. 1. Playability as characterisation and measurement of the interaction experience

- ***Satisfaction:*** It is defined as the gratification or pleasure derived from playing a complete video game or from some aspect of it such as mechanics, graphics, user interface, story, and so on. Satisfaction is a highly subjective attribute that is by definition difficult to measure as it depends on the preferences of each player, which in turn influence the satisfaction derived from specific elements of the game (characters, virtual world, and challenges).
- ***Learnability:*** It is defined as the player's capacity to understand and master the game's system and mechanics (objectives, rules, how to interact with the video game, and so on). Desktop Systems try to minimize the learning effort, but in video games we can play with the 'learning curve' according to the nature of the game. For example, on the one hand, a game may demand a high initial skill level before playing, or it may put the player on a steep learning curve during the first phases of the game, to help them understand and master all the game's rules and resources so that they can use them virtually from the outset. On the other hand, players can learn step-by-step in a guided fashion when they need to develop a particular ability in the video game.
- ***Effectiveness:*** It is defined as the time and resources necessary to offer players an entertaining experience whilst they achieve the game's various objectives and reach the final goal. An 'Effective' video game is able to engage the player's attention from the outset through to the very end of the game. Effectiveness can be analysed as the correct use of the challenges by the player throughout the game, the correct structuring of the objectives of the game and/or the best adaptation of the controls to the actions in the game.
- ***Immersion:*** It is defined as the capacity of the video game contents to be believable, such that the player becomes directly involved in the virtual game world. This intense involvement means that the player effectively becomes part of the virtual world, interacting with it and with the laws and rules that characterize it. A video game has a good Immersion level when it achieves a balance between the challenges it presents and the player abilities necessary to overcome them.

- ***Motivation:*** This is defined as the set of game characteristics that prompt a player to realize specific actions and continue undertaking them until they are completed. To achieve a high degree of Motivation, the game should offer a set of resources to ensure the player's perseverance in the actions performed to overcome challenges. By 'resources' we mean different elements to ensure positive behaviour in the interpretation of the game process, thereby focusing the player on the proposed challenges and their respective rewards, showing the relevance of the objectives to be achieved, and encouraging the player's confidence and pleasure in meeting and achieving challenges.
- ***Emotion:*** This refers to the player's involuntary impulse in response to the stimulus of the video game that induces feelings or a chain reaction of automatic behaviours. The use of Emotion in video games helps achieve an optimum Player Experience by leading players to enter different emotional states.
- ***Socialization***: It is defined as the set of game attributes, elements, and resources that promote the social dimension of the game experience in a group scenario. This kind of collective experience makes players appreciate the game in a different way, thanks to the relationships that are established with other players (or with other characters from the game). Game Socialization allows players to have a totally different game experience when they play with others and it promotes new social relationships thanks to interaction among players. Socialization is also at work in the connections that players make with the characters of the video game. Examples of this might include: choosing a character to relate to or to share something with; interacting with characters to obtain information, ask for help, or negotiate for some items; and how our influence on other characters may benefit, or not, the achievement of particular objectives. To promote the 'social factor', new shared challenges need to be developed that help players join in with and assimilate the new game dynamic, creating a set of collective emotions where players (or characters) encourage and motivate themselves and each other to overcome collective challenges.

Playability analysis is a very complex process due to the different perspectives that we can use to analyse the various parts of a video game architecture. In [6] a classification of these perspectives is based on six Facets of Playability. Each facet allows us to identify the different attributes and properties of Playability that are affected by the different elements of video game architecture [7]. These Facets are:

- ***Intrinsic Playability:*** this is the Playability inherent in the nature of the video game itself and how it is presented to the player. It is closely related to Gameplay design and Game Mechanic.
- ***Mechanical Playability:*** this is the facet related to the quality of the video game as a software system. It is associated with the Game Engine.
- ***Interactive Playability:*** this facet is associated with player interaction and video game user interface development. This aspect of Playability is strongly connected to the Game Interface.
- ***Artistic Playability:*** this facet relates to the quality of the artistic and aesthetic rendering in the game elements and how these elements are executed in the video game.

- ***Intrapersonal Playability*** or Personal Playability: This refers to the individual outlook, perceptions and feelings that the video game produces in each player when they play, and as such has a high subjective value.
- ***Interpersonal Playability*** or Social Playability: This refers to the feelings and perceptions of users, and the group awareness that arises when a game is played in company, be it in a competitive, cooperative or collaborative way.

The overall Playability of a video game, then, is the sum total of values across all attributes in the different Facets of Playability. It is crucial to optimise Playability across the different facets in order to guarantee the best Player Experience.

4 Video Game Test and Experience Evaluation based on Playability

We recommend evaluating and testing the Playability and Player Experience during the entire development process in order to ensure the quality of Playability/Experience in every playable video game element or game content in the final product. When testing or evaluating experience, a Playability Model is used to achieve the following objectives: analyse the player experience in a quantitative/qualitative way using the playability model; test the effects of certain elements of a video game on overall player experience; identify problems that may cause a negative player experience; complete the functional assessment and objectives of QA systems with non-functional evaluations that are closer to the experience for each player profile; and offer reports that provide complete information of every aspect of the player experience.
In typical Playability and PX evaluation there are four steps to test pragmatic and hedonic attributes of the experience [24] see Fig. 2: ***Pre-Test***, questionnaires and a short test to obtain information about player profiles; ***Test***, to collect information in real time about player experience while users play a video game; ***Post-Test***, players are given different questionnaires and interviews to be completed, particularly, with subjective information related to hedonic properties. The questionnaires should be guided by the Facets of Playability; and ***Reports***, to obtain a number of reports about the PX including information about which playability attributes have more influence, or which type of elements are more valued by players.

Fig. 2. Playability Evaluation Phases for video games and entertainment systems

In this point we present a practical example of haw to use Playability to analyze the interactive experience with a real video game: “Castlevania: “Lords of Shadows” [25]. This game was developed by the Spanish company MercurySteam and published by Konami. The game is a reboot of the franchise. It is an action-adventure game in a dark fantasy (horror/fantasy) setting in Southern Europe during the Middle Ages (in the year 1047) for PlayStation 3 and Xbox360 consoles. One objective of the work is to develop the UX evaluation to extract results *easily* and *cost-effectively* for the company and is effortlessly adaptable to the traditional quality assurance that video game developers perform with the product.

The experiment involved the participation of 35 student volunteers from different degree courses students at the University of Lleida, Spain. The students did not know about Castlevania. The evaluation was conducted at the UsabiliLAB, the usability laboratory of the Universitat de Lleida. The evaluation equipment was based on two computers. One of them was for the user and it was equipped with the Morae Recorder, which registered user interaction, screen video, user interactions, user voice and video through a web-cam. It is highly advisable to work with different player profiles (game stakeholders) so that the results are representative of the context of real-life video game use. The majority of the participants (*Pre-Test* information and results) were male (75%) between 19 and 22 years old (90%). They were considered to be casual players (playing approximately 5 hours per week, and with experience of only one games console or a mobile phone). They had knowledge of different gaming platforms, including a mobile and a desktop platform (90%). The preferences for different game genres were: adventure (≈60%) and fighting (≈30%). 87% preferred to play in company.

During the *Test*, the users play the beta of the videogame (the first two levels, *Test Cases*, Figure 1) at a ‘normal’ difficulty level. The goal of the first level is to save a village from wolves and wargs. The goal of the second level is to run away from wargs and fight against them using a horse and finally escape from them. Using Morae software and paper emotional script, evaluators tag expressions to analyse later in the making of the reports. The objective is to identify looks and facial expressions with the oral perception of the video game. To perform the gameplay analysis we use Playability Metrics [6]. This metrics offer information about properties and attributes of Playability (see previous points). These metrics are measured with observation techniques such as those indicated in this work. These metrics are contextualised by the mechanical and dynamical nature of this video game.

In the *Post-Test*, informal interviews and questionnaires are used to obtain information about the player experience. The evaluation process is performed using a list of questions, with the aim of investigating the degree of each attribute of Playability and Facets in order to obtain a measurement of the player experience. We readapt the list of questions from validated works [3, 5, 6, 12, 13, 14, 20, 21]. The rating scale for the answer is 1 (very unsatisfied) to 5 (very satisfied). Each question is related to a facet and property of Playability.

The analysis of the experience performed by Playability measurements (Table 1 and Figure 4) shows that this game causes a very balanced UX, as there is not attribute or facet that is highlighted over others. The average punctuation in questionnaires marks a value near to 4, which represents a video game with a good value of interactive experience for players.

Table 1. Questionnaire and results to analyse PX guided by Facets of Playability

Questions	Max	Min	Avg.	Std. Dev.
Intrinsic Playability				
1. The way you play the game is fun and interesting	5	2	3,84	0,80
2. Key concepts to play are easy to remember and learn	5	3	4,22	0,60
3. The game has offered fun from the moment you started with it	5	2	3,45	1,25
4. The elements of the game are consistent with the game story	5	3	4,15	0,70
5. The game provokes the need to keep playing to know what will happen with the protagonist	5	2	4,00	1,00
6. Game elements successfully transmits the emotions of the game dynamics	5	2	3,65	1,05
7. The game allows you to interact with other characters and use different elements that they offer	4	0	2,74	1,25
Avg.	**4,86**	**1,57**	**3,72**	**0,95**
Mechanical Playability				
1. The game engine exploits the graphic and physical characteristics of the virtual world	5	2	3,81	0,85
2. The game offers a dynamic and context-sensitive help for the current challenges	5	3	3,91	0,80
3. The game provides mechanisms to capture the player's attention	5	2	3,69	1,05
4. The lighting and rain effects are similar to reality	5	3	4,19	0,95
5. The game makes it easy to learn new moves and improvements	4	3	3,95	0,85
6. The Facial expressions and gestures are understandable and representative to the context of game	5	3	3,65	0,90
7. The Social interaction and dialogue with other characters is fluid and natural	5	0	2,95	1,40
Avg.	**4,86**	**1,86**	**3,73**	**0,97**
Interactive Playability				
1. The control system is balanced and easy to interact.	5	3	4,10	0,70
2. Learn the key combination is easy to remember	5	2	4,05	1,00
3. To Access and use actions and secondary armament is fast and fun.	4	2	3,36	1,10
4. The game interface is not intrusive, and it is natural to the nature of virtual world	5	3	3,69	0,70
5. The game's information helps you to continue playing	5	3	4,15	0,70
6. The storytelling helps you understand how the protagonist feels	5	2	3,24	1,15
7. The Interaction with other characters is entertaining and relevant to the game process	4	2	2,98	0,85
Avg.	**4,71**	**1,93**	**3,65**	**0,89**
Artistical Playability				
1. The story and narrative are interesting	5	0	3,48	1,45
2. The cut scenes and movies have a high degree of quality	5	3	4,00	0,85
3. The game's visual elements are recognizable elements of familiar places or monuments of humanity	5	2	3,91	0,95
4. The music is appropriate to the action and game dynamics	5	3	3,98	0,75
5. The game does not discover story elements that may affect the future interest of the player	4	2	3,38	0,85
6. The artistic elements transmit emotion to the player	5	3	3,81	0,95
7. Being an ally or enemy is easily identifiable during play	5	2	4,34	1,10
Avg.	**4,86**	**1,71**	**3,84**	**0,99**
Personal Playability				
1. The obtained fun when playing is adequate	5	2	3,81	1,05
2. The entertainment was appropriate to the game duration	5	2	3,79	0,95
3. The difficulty level was suitable	4	1	3,74	0,90
4. Accuracy and the skill to preform the actions was right	5	3	3,79	0,80
5. The sound effects helped to continue the game action	5	3	4,31	0,90
6. The nerves did not affect the way you played	5	0	3,53	1,40
7. I would have liked to share the game with friends	5	0	3,60	1,50
Avg.	**4,86**	**1,29**	**3,79**	**1,07**
Social Playability				
1. New game objectives, rules and challenges are easily identified when several players play the game.	4	1	2,77	1,32
2. Social interaction helps to improve the game actions	5	1	3,53	0,95
3. The social interaction helps to understands and feels the story	5	0	2,12	1,45
4. There are game elements to identify the identity of each player with the virtual world.	4	0	1,8	1,23
5. The social game players or controls with other characters, differ from the individual game system.	4	0	2,24	0,89
Avg.	**4,40**	**0,40**	**2,49**	**1,17**
TOTAL Avg.	***4,83***	***1,67***	***3,75***	***0,97***

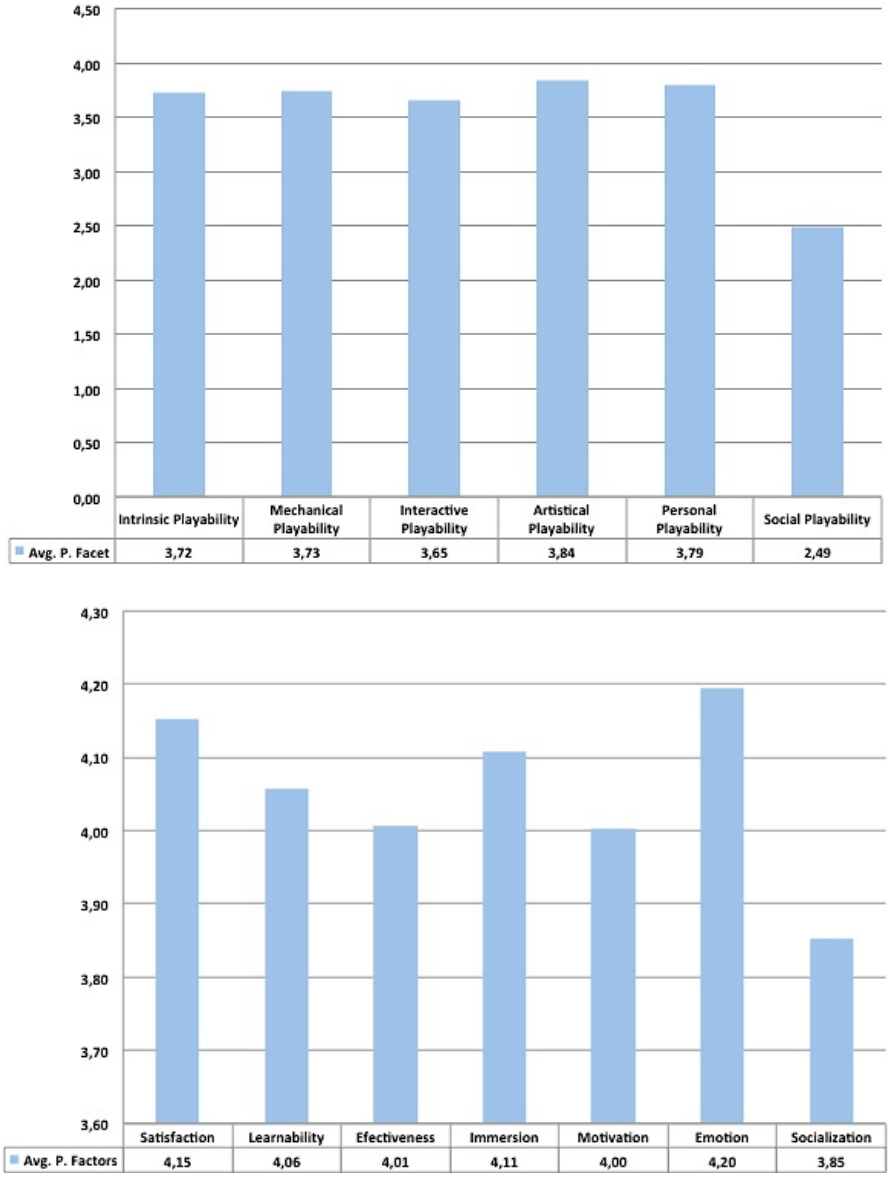


Fig. 4. Playability values for "Castlevania: Lords of Shadow"

Notably, the central user comments focused on high quality graphics and cinematic effects (Artistic and Mechanical properties) that caused the desire to continue playing (Satisfaction-Motivation-Immersion). Moreover, the technical quality of the game, together with concern for good artistic effects causes the Motivation and Immersion to be high. The ease of controlling and using the game interfaces to perform the

actions (or game combos) in the video game and overcome the goals is an incitement to play. This can be corroborated thanks to Interactive Facet.

Also, in Learnability; the punctuation could be better if the video game were to incorporate more adaptation facilities. The socialisation option also has a good value due to the 'number of social elements' that players used, but could be better. The Personalisation (effectiveness and motivations properties) level is low due to these being the first levels of the video game, and therefore players cannot sufficiently personalise the weapons or combos with which to attack the enemies. Everybody remarks on the capacity of the video game to excite and to 'fight' energetically with the enemies (emotion-immersion-satisfaction).

The *Post-Test* results reaffirm the good opinion of the users as to the localisation and game context (immersion), as well as to sound effects and music in general (high level of satisfaction, mainly characterised by a state of pleasure, which varies between excitation and neutral state (immersion-emotion). These factors contribute to a better atmosphere and emotion, emphasising the closeness and familiar culture players chose for the various elements of the game based on Lovecraftian ideas.

Globally, the UX analysis indicates that the results pertaining to interactive experience are very positive because most players (experts or casual players) played equally, see Table 1 (Avg. and Std. Dev. values).

5 Conclusions and Future Work

This work reflects the importance of analysing the experience that players have with video games in a pragmatic and hedonic way. Understanding and evaluating the user experience in video games is important for developing more efficient and successful products in terms of entertainment. Our work reflects the importance of having a playability measurement model to analyse a player's experience of the process of use of a videogame. In our proposal, the characterisation of playability is related to the evaluation of player satisfaction and the evaluation of the effect of video games.

We have presented the concept of Playability as a crucial characteristic of Player Experience in video games, outlining the attributes and properties that characterise it in order to measure and guarantee an optimum Player Experience. To facilitate the analysis of Playability, we have proposed six Facets of Playability in order to study every property in each attribute and identify the elements necessary to achieve a positive overall Playability in different video games. Our proposal have been applied in the case of "Castlevania: Lords of Shadow" and are promoted by the Academy of Interactive Arts and Sciences of Spain.

Now we are working on incorporating specific metrics, questionnaires and heuristics from other research projects (see previous point) and public methods from different video game development companies and QA studios to extend our proposal and have enough alternatives for different player profiles, video game genres and platforms in the video game industry to perform a more complete and unified analysis of video games. Results of our research project are a complementary alternative to the traditional tests and methodological QA evaluations performed by the video game industry professionals of each company. This proposal will always be in progress

according to the requirements of the studios and video game developers and market. Playability Model is not as complete as other specific UX techniques for a particular attribute or measurement criteria. But the model is open to the inclusion of more properties, criteria, metrics, indicators or techniques to improve the model to characterise the quality of the interaction and the experience of video games; this is part of our current work.

Finally, we are also currently updating and completing the different development phases of the Player Centred Video Game Development Process, particularly the requirement analysis and evaluation of experience. We are working on including the use of agile methodology to help us iterate on different game prototypes, we are incorporating playability factors to guarantee the experience in order to evaluate and improve playable prototypes and readapt the user experience to the possible changes in requirements and preferences which can occur when players test the game. We are integrating these ideas in educational and sanitary rehabilitation video games where a positive experience is a crucial factor for the success of the video games as support tools.

Acknowledgments. This work is financed by the Ministry of Science & Innovation, Spain, as part of VIDECO Project (TIN2011-26928). We wish to thank MercurySteam for the multimedia material, and the students of the University of Lleida (Spain) for their support in this work.

References

1. ISO/IEC 9241-210.: Ergonomics of Human–System Interaction – Part 210: Human centred Design for Interactive Systems. Clausule 2.15 (2010).
2. Law, E. et al.: Understanding, Scoping and Defining User Experience: A Survey Approach. Proc. Human Factors in Computing Systems (CHI'09), pp.719-728. (2009).
3. Bernhaupt, R (eds).: Evaluating User Experience in Games: Concepts and Methods. Springer. (2010).
4. Nacke, L et al.: Playability and Player Experience Research. DiGRA, Breaking New Ground: Innovation in Games, Play, Practice and Theory (2009).
5. Isbister, K., Schaffer, N (eds).: Game Usability: Advancing the Player Experience. Morgan Kaufmann (2008).
6. González Sánchez, J. L.: Jugabilidad y Videojuegos. Análisis y Diseño de la Experiencia del Jugador en Sistemas Interactivos de Ocio Electrónico. Ed. Académica Española, Lambert Academic Publishing GmbH & Co KG. (2011).
7. Rollings, A., Morris, D.: Game Architecture and Design. New Riders Games (2003).
8. Swink, S.: Game Feel: The Secret Ingredient. Gamasutra. http://www.gamasutra.com/view/feature/2322/game_feel_the_secret_ingredient.php [Accessed 5 May 2013].
9. Saito, A.: Gamenics and its potential. In Isbister, K., Schaffer, N. (eds) Game Usability: Advancing the Player. Morgan Kaufmann (2008).
10. Lazzaro, M. The Four Fun Key. In Isbister, K., Schaffer, N. (eds) Game Usability: Advancing the Player. Morgan Kaufmann (2008).
11. Järvien, A. et al. 2002. Communication and Community in Digital Entertaiment Services. Hypermedia Laboratory Net (2). http://tampub.uta.fi/tup/951-44-5432-4.pdf [Accessed 07 May 2013].

12. Federoff, M.: Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Video Games. (Master of Science Thesis). Indiana University (2002).
13. Desurvire, H., Wiberg, C.: Game Usability Heuristics (PLAY) for Evaluating and Designing Better Games: The Next Iteration. In Ozok, A.; Zaphiris, P. (eds). Proceedings of the 3d International Conference on Online Communities and Social Computing: Held as Part of HCI International 2009 (OCSC '09). Springer-Verlag, Berlin, Heidelberg, pp. 557-566 (2009).
14. Pinelle, D., Wong, N., Stach, T.: Heuristic evaluation for games: usability principles for video game design. In Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (CHI '08). ACM, New York, NY, USA, pp. 1453-1462 (2008).
15. van den Hoogen, W.M., IJsselsteijn, W.A., de Kort, Y.A.W.: Exploring Behavioral Expressions of Player Experience in Digital Games. In Proceedings of the Workshop on Facial and Bodily Expression for Control and Adaptation of Games ECAG 2008, pp. 11-19 (2008).
16. Canossa, A., Drachen, A.: Play-Personas: Behaviours and Belief Systems. In User-Centred Game Design. Human-Computer Interaction - Interact 2009, Pt II, Proceedings, Volume 5727, pp. 510-523 (2009).
17. Björk, S., Lundgren, S., Holopainen, J.: Game Design Patterns. In Proceedings of Level Up 1st International Digital Games Research Conference (2003).
18. Chen, M., Cuddihy, E., Thayer, A., and Zhou.: Creating cross-cultural appeal in digital games: Issues in localization and user testing. In Presentation at the 52nd annual conference for the Society for Technical Communication (STC). http://markdangerchen.net/pubs/Game_Slides_final3.ppt [Accessed 11 May 2013] (2005)
19. Padilla Zea, N., Gonzalez Sanchez, J.L., Gutierrez Vela, F.L., Cabrera, M., Paderewski, P.: Design of educational multiplayer videogames: A vision from collaborative learning, In Advances in Engineering Software, 40 (12), 1251-1260 (2009).
20. Poels, K., Kort, IJsselsteijn, W.A., de Kort, Y.A.W.: Development of the Kids Game Experience Questionnaire. Poster presented at the Meaningful Play Conference, East Lansing, USA (2008).
21. Poels, K., IJsselsteijn, W.A., de Kort, Y.A.W., Van Iersel, B.: Digital Games, the Aftermath. Qualitative insights into Post Game Experiences. In R. Bernhaupt, R. (eds.). Evaluating User Experiences in Games. Berlin: Springer, pp. 149-165 (2010).
22. Westin, T., Bierre, K., Gramenos, D., Hinn, M.: Advances in Game Accessibility from 2005 to 2010. In Stephanidis, C. (eds) Universal Access in Human-Computer Interaction. Users Diversity. Lecture Notes in Computer Science 6766, pp. 400-409 (2011).
23. Yuan, B., Folmer, E., Harris, F. : Game accessibility: a survey. In Universal Access in the Information Society. 10, 81-100 (2010).
24. González Sánchez, J. L., Gil Iranzo, R. M., Gutiérrez Vela, F. L.: Enriching evaluation in video games. In: Campos, P., Graham, N., Jorge, J.A., Nunes, N., Palanque, P., Winckler, M. (eds.), Human-Computer Interaction - INTERACT 2011 - 13th IFIP TC 13 International Conference Proceedings, Part IV. LNCS, vol, 6949, pp. 519-522, Springer, London (2011).
25. Castlevania: Lords of Shadow: http://www.konami.com/officialsites/castlevania/

Creativity and Entertainment: Experiences and Future Challenges

Alejandro Catala, Javier Jaen, Patricia Pons, Fernando Garcia-Sanjuan

Grupo ISSI
Departamento de Sistemas Informático y Computación
Universitat Politècnica de València
Camino de Vera S/N
46022 Valencia (Spain)

{acatala,fjaen,ppons,fegarcia}@dsic.upv.es

Abstract. This paper revisits the idea of involving creativity along entertainment. The review of the related work shows that the creation of artifacts is usually directed to programming simulations and the creation of entities to be involved during the play. It is observed that proposals providing more creative capabilities are usually based on WIMP interfaces, negatively impacting on the role of collaboration and active participation. This paper presents the main findings of the CreateWorlds project, which relies on an interactive tabletop interface in order to support the creation of digital ecosystems, and states several challenges that will have to be addressed to build future creative game-based learning environments.

Keywords: creativity, entertainment, challenges, ambient intelligence, user engagement, learning

1 Introduction

Supranational entities such as the EU commission are recognizing creativity as a key driver for economic development in a competitive global market. The EU is fostering the development of a knowledge and creative based society by means of the strategic framework for European cooperation in education and training [1]. It aims at enhancing creativity and innovation at all levels of education and training, a response to facilitate the change in our society and the economy in the long term.

Although the term creativity seems always to cause controversy due to the wide range of definitions [2], all of them essentially refer to the idea of novelty, originality, innovation, unusual, different, etc. Creativity is defined in most of the main thinking streams as a multidimensional concept, which is related to several personal traits, skills, product features, processes, and environmental conditions [3].

In one way or another, creativity is important because it is related to divergent thinking and the process of generating and exploring multiples ideas to obtain reasonable solutions to complex or open problems and face new challenges.

Digital information and communication technologies (ICT) have been seen as powerful tools to facilitate learning in general, as well as to develop creativity and related skills. The main features behind this idea are the exploration and representation of ideas in multiple ways (multimedia); interactivity; reproducibility of problems in a controlled manner; retrial without or with a reduced risk in case of fail; and communication and information sharing among users for discussion.

Myriads of digital systems have been created with learning purposes and to support learning tasks [4], [5], [6]. They have been used as a complement in informal learning settings and relied on many different technologies: from multimedia offline applications on CD/DVD to online web-based on desktop systems; from virtual and augmented reality in static settings to networked augmented and mobile systems, etc. Regardless of the base technology, learning activities have been focused on games and playing activities and have relied on appealing and engaging user interfaces. This shows that entertainment along with constructivist approaches as a resource are used as a mean to facilitate learning [7]. The point is that learning would take place if these systems succeeded in keeping users highly motivated and engaged in the activities for a sustained period of time.

This paper discusses, in section 2, how a set of learning systems focused on creativity have been used for learning purposes, and section 3 presents our own research based on digital tabletops and discusses the related experiences. Section 4 poses some general challenges that must be taken into account by those systems, including entertainment oriented, that use new ICT with the purpose of supporting some kind of creative learning.

2 Related Work

The present section briefly reviews, in a non-extensive way, a collection of selected works that have used digital technologies in creative tasks with the purpose on learning. For two decades, these proposals based on entertainment technology for creative learning have focused on the creation of some kind of artifacts. According to the supported creation capabilities, the reviewed works fall within two primary categories identified here.

2.1 Programming and Performing in a Pre-established World

This first category deals with those proposals that have been focused on supporting performances of pre-established characters or entities in a world ecosystem in a broad sense. Hence, the common feature for works in this group is that users cannot create the characters or entities themselves but these are already pre-existing and then users are allowed to perform with them or specify their behavior by encoding a program to tell a story.

Since the programming language Logo and Graphics Turtle [8] was devised as a way to show and teach computational concepts, many proposals have been inspired in a subset of the features of Logo and have focused on social and collaboration skills.

For instance, AlgoBlock [9] describes an educational tool where users use physical block-like pieces that can be arranged all together to program the movement of a submarine within a labyrinth. The result of the program execution is shown on a monitor by means of an animated submarine moving on a map. The system allows students to improve their skills in problem-solving by means of some sort of collaborative programming tasks.

A selected second example is Cleogo [10]. It is a programming environment for groups. The aim is to encourage children to solve problems collaboratively. Each user has a screen equipped with a keyboard and mouse, but the input controllers provide access to all the functionality of a shared widget displayed in the different screens.

A third outstanding example using Logo is Turtan [11]. It is a tangible programming language that uses a tangible surface interface in which tangible pieces represent virtual instructions of programs. As in the original Logo language, one of the design objectives of TurTan is learning programming concepts. The work mentions the necessity to explore learning and creativity as the system is oriented to young children and because its potential to support the artistic expression and collaborative scenarios.

Another relevant work is the exploration on tangibles carried out in [5]. They discussed a system that allows children the creation, edition and simulation of a 2D virtual environment in a collaborative way. The TangibleSpaces system consists of a large carpet with an array, a set of plastic cards with RFID tags representing entities and operations, and a screen which shows the state of the system under construction.

Instead, Quetzal and Tern are tangible languages to specify the behavior of robotic ecosystems [12]. They use physical objects (plastic or wooden pieces) with no electronic device but with visual tags. These languages were designed to teach basic programming to children in a classroom setting in primary and secondary schools. The robots can interact in the same world, and several student groups can collaborate in a shared activity to solve problems to pick up objects or navigate through a labyrinth.

IncreTable represents a great effort on joining tangible and interactive digital interfaces to foster creativity [13]. It is a game based on The Incredible Machine that uses an advanced high technology setting focusing on mixed reality tabletops. The game consists of several puzzle exercises that require the construction of Rube-Goldberg machines, involving virtual as well as actual domino pieces and other physical objects such as portals or robots. Levels are supposed to encourage user creativity to solve the puzzle in a complex way.

Finally, StoryTelling Alice [14] is a programming environment aimed at girl teenagers to encourage them to learn programming skills. The girls can tell their stories in the virtual environment by means of programs. They construct free of syntax errors by simply drag&drop interaction techniques and through the pre-visualization mode they can see the resulting animation to check whether the instructions do what they wanted for their stories.

The main commonality of all these works is their focus on the creation of procedural behavior for virtual entities, but the creation of the world ecosystem is less common. In addition, most of them consider co-located activities mainly mediated by tangible interaction.

2.2 Creating Characters for an Ecosystem World

This second category includes those proposals that have considered the creation of characters/entities as a central task. Once characters are created, most of these works rely on a computational model to specify the behavior of these entities to build the simulation or the story.

LogoBlocks is a graphical programming language to support programming for the LEGO programmable brick [15]. The brick is a small computer that can be embedded and used in LEGO creations by reading from sensors and controlling engine activations. LogoBlocks follows a drag&drop methaphor in a WIMP user interface. Although LogoBlocks itself is simply the graphical programming language, it is targeted at the programmable brick and therefore the constructions of the robots are creations that can also be used as part of an ecosystem.

Scratch [4] is a graphical programming environment that allows children to program interactive stories, games, animations and simulations. All these are based on 2D stages composed of a background and a set of sprite-based objects and the language used is based on LogoBlocks. There exists a web-based online community which aims to foster discussion and creativity between users, relaying on collaboration, mutual discussion and distributed contribution.

Another relevant work with the idea of creating 2D entity-based virtual ecosystems in full is AgentSheets [16]. It allows users to create simulations and 2D interactive games based on a rectangular array. The users have to design the visual aspect of agents by drawing icons and specify their behavior using event-based visual rules.

An innovative tangible approach is Topobo [6]. It is a 3D constructive assembly system with kinetic memory that allows the creation of biomorphic forms like animals and skeletons. It is designed to be a user interface that encourages creativity, discovery and learning through active experimentation with the system.

ShadowStory [17] presents a storytelling system inspired in Chinese traditional shadow puppetry. Children use a laptop with touch input to create digital animated characters and other accessories or props, and then they are allowed to perform stories on a back-illuminated screen, controlling the characters with simple movements by means of orientation handheld sensors.

The works described in this category focus on the creation of artifacts, such as the entities, the entity behavior, and the world ecosystem itself. However, in most of the cases they accomplish this by using a Personal Computer (PC), what makes them less suitable to support collaborative tasks. As such social interaction is also an important part in developing creativity, our running project CreateWorlds attempts to bring a proposal in which co-operation and collaboration is central.

3 CreateWorlds: Experiences on Creating Ecosystems

3.1 Vision

Constructivist methods are usually present in videogames and interactive media for learning. However, the works in the previous section are going further because they

support the creation of artifacts in several ways so that learning is triggered by completing creative tasks. It means that they do not only require users to adopt an active role and carry out some actions to achieve goals while consuming media. In addition, they require creating something new, resulting in extra cognitive effort and supposedly in more rewarding experiences in terms of learning.

In our opinion, future gaming platforms supporting learning activities should provide pre-set scenarios easily editable by teachers to adapt them to the curriculum content, and a space for experimentation, discussion and reflection for both teachers and learners. Therefore, future research on serious games for learning should not be focused on producing games whose content fits to a wide range of educational topics and learning styles but instead on giving teachers and learners the tools to produce their own games. Under this vision we have run the project Createworlds that contributes to the field of creativity and entertainment in several ways.

On the one hand, it aims at flexibly supporting discussion, action and reflection processes in a similar way as considered by educators and researchers when combining traditional teaching materials with videogames [18]. This is especially important to strengthen attitudes and develop certain skills related to creativity, such as generating ideas, set and testing hypothesis, collective discussion, exchange of ideas, curiosity, independence, novelty, flexibility, openness to new experiences, etc.

On the other hand, it brings the pedagogy posed by Clark Abt at 70s, when he set the creation of traditional play as an effective mean to creative learning [19]. The idea is to consider the creation of games and the subsequent play as the creative task to be carried out, not limited to a very specific part as happened in the reviewed works. It entails, as he discussed, that students take the role of game designers, who are actually inventing a simulation model of the process to be played. In the course of doing so, the student must identify the significant variables involved, the relationships among them, and the dynamics of the interaction. To do this successfully, it requires understanding the process to be simulated. Therefore, involving the students in this process expands their knowledge, learning not just factual content but also the processes and interactions involved.

3.2 Realization

The project has been realized by means of several prototypes relying on tabletop technology. Interactive tabletops are gaining increased attention and promising to success in education [20]. The decision to use digital interactive surfaces is motivated by the priority of issues to fulfill the vision, and the idea of following more natural interaction platforms, facilitating the discussion and collaboration among participants, and therefore not being the technology a significant barrier but a medium to enhance the overall experience.

In this way, the interaction is face-to-face supported by both verbal and non-verbal communication (i.e. speech and gestural) in a small group of participants, without technological barriers as in networked approaches. Therefore, the mediation entirely relies on social protocols for negotiation and collaboration. In turn, the limitation is that activities must be always co-located, there is no a record of communications, and

the number of participants per session is lower than in networked scenarios. The barrier that keyboards and peripheral devices usually introduce has been removed thanks to the interaction supported by using fingers and tangible tokens on the digital surface. This interaction paradigm is expected to be more attractive to start interactions as it is more natural, easier, and its penetration in the mass market of smartphones and tablets.

CreateWorlds supports the creation of game ecosystems where 2D virtual entities can be located and simulated according to intelligible physics (i.e. kinematics). The ecosystem world primarily supports the creation of entities and the specification of their behavior as reactive rules. The entities, instead of being simply represented by single bitmaps or sprites as in most 2D game tools, we decided to support the definition of entities composed of architectural units in the same way that a skeleton is composed of bones. In this way, we can also support the creation of more advanced behaviors and the more interesting learning scenarios based on storytelling containing puppets [21].

In most cases, games require much more than the kinematic simulation and need to control the change of value properties under certain situations. A model of reactive rules have also implemented in CreateWorlds. This model considers the condition and the action as expressions so that they can be visually represented by dataflows [22]. It has facilitated a rule editor that supports the composition of dataflows by connecting lines with fingers between operators selected by touch drag&drop techniques [23].

Hence, the editors for the creation of both entities and rules have been designed as compositional tasks that consist of assembling components. At any moment, the ecosystem player can be called. It interprets the edited artifacts and starts an interactive enactment combining entity simulation with rules behavior [21]. Users can check whether the game ecosystem is running as expected, interact directly with entities through their fingers, and eventually modify and fix issues they find. It facilitates quick trials and execution to improve the ecosystem incremental and iteratively. Figure 1 shows different stages at using the prototypes.

The platform can save the ecosystem as users progress in their tasks in order to retrieve the creative sessions later. It also allows loading pre-set scenarios containing partial game ecosystems. It is particularly useful for teachers and tutors to design learning sessions in which participants are assigned to complete a specific part of the ecosystem according to specific learning goals.

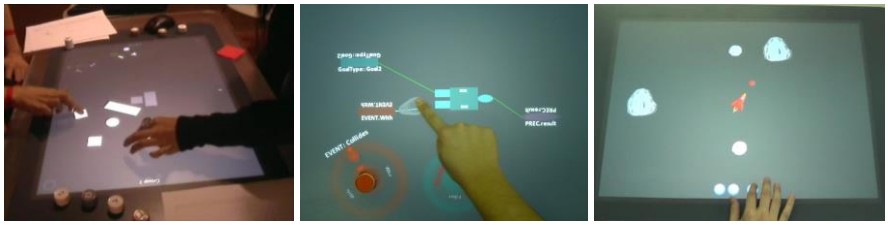


Fig. 1. (Left) Users creating an entity-based simulation; (Center) Editing a dataflow rule; (Right) Playing an asteroids game running on the middleware.

3.3 Main Findings

A simplified version of the entity editors has been used in several empirical studies to test the thinking-discussion-reflection loop in creative tasks. Teenagers were proposed two different tasks, namely freely creation of entity-like figures [24], [25], and creation of Rube-Goldberg machines [26].

In these sessions we observed how direct communication and manipulation on tabletops is carried out. The digital surface encourages participation and keeps participants motivated, confirming that the target technology is suitable for our purpose. The interface invites users to touch unconcerned, try and experience variations of their creations until they get the ecosystem working properly. Such iterative retrial refinement is positive as it allows to test ideas without fear for failure. Fails must be taken as new inputs for later success and as necessary events in the path to learning.

The surface can be seen as a canvas in which assembling components remain, and where including and manipulating more digital items have a low cost for users. Therefore, it facilitates the exploration of new ways to connect the entity components and users are able to elaborate and explore more complex ideas in the digital ecosystem like this than those they would be allowed with traditional learning materials.

It is observed that the surface helps in sharing the digital objects. It is also a positive point as participants should actively participate in the construction of the ecosystem. Among the several manipulation patterns observed, a fair co-operation pattern prevailed. It is worth to mention that all this has been built upon a consistent and homogeneous exploration framework provided by Tangiwheel [27], which is essentially a flexible widget for collections designed to support simultaneous parallel explorations in multi-user surface applications. During the experimental sessions, we embedded the thinking-discussion-reflection loop so that teenagers had to work in groups in activities that combine paper and pencil materials with the use of the digital system. This combination is also positive since it ensures that teenagers actually prepare their ideas and discusses them with the partners before implementing them on the digital surface. This is in favor of creative and social skills that we are interested in developing on users. Finally, such a loop showed that the surface can effectively be used to split the learning sessions in different ways, so that tutors can design the base materials to be provided and the participants can keep their progress between sessions.

Additionally, we have conducted a preliminary evaluation on the comprehension of the dataflow expressions to be used in the rule editor [22]. The study involved teenagers at secondary schools, and it showed that the visual language for dataflow is preferred and that young non-expert programmers are able to understand and even modify the expressions without tool support.

4 Future Challenges: Towards Creative Smart Learning Environments

Despite the previous technological efforts to support creative learning, we believe there are still serious challenges that need to be addressed in the field of entertainment

or game experiences that enable creative learning scenarios which we will be briefly discussed in the following.

4.1 Real versus Virtual Experiences

All the previous approaches, including CreateWorlds, support the creation of ecosystems that exist in a simulated way. However, new paradigms such as ubiquitous computing bring new possibilities for new creative learning game experiences based on reactive entities that not only exist in a virtual world but which are also part of the surrounding spaces. In this sense, it would be reasonable to think that smart environments may be effective tools to support creative learning gaming experiences.

4.2 Tools versus Applications

Thinking and learning processes are plastic and flexible, as they can change dynamically under different circumstances. The ways which teachers teach and students learn are diverse, as well as the learning activities and tasks are. Thus, there are very important sources of variability to deal with, which would lead to complex software requirements and advanced architecture design. However, digital learning systems are mostly focused on a series of activities to be carried out and completed in a very specific order, resulting inflexible.

In the same way that learning materials are prepared and developed for use in traditional learning environments so that they suit the curriculum contents and the planned sessions flexibly, future learning environment supported by ICT should also manage and facilitate this complexity. The point is that as any other software, if workflow requirements are not met (e.g. the way users would like to use it within their ordinary processes), it will be useless and the interest in adopting it would therefore drop.

Therefore, a very important challenge is that of tools supporting the creation of materials and the management of learning sessions rather than inflexible ICT applications supporting a very specific task. A sample that shows how important is fitting such variability in the processes is the effort in studying the weaving of mass videogames and computer software not designed for learning with reflection and discussion processes and other traditional tasks in learning environments [18]. It shows a perspective that seems more cost-effective, adaptable and especially it shows that ICT must remain as a secondary but important element behind learning purposes.

4.3 Natural and Tangible Interfaces

For sure one of the most important efforts to be made is that of interfaces, since they are the visible part and the entry point to the systems. It is important that user interfaces invite users to interact with them and they must be easy to use and manipulate. In general, interfaces in the entertainment industry are attractive and highly interactive, providing quick response to actions and displaying information with immediacy. These interfaces are more focused on responding to actions and the consumption of information in form of multimedia experiences.

Nevertheless, when artifact creation is needed, interface design requires more attention and it is more complex. The point is that ordinary and mass controllers are very effective when used to navigate spaces, consume linear information, select short options or input immediate short actions, but they weaken when they need fitting to longer input processes as in creative/creation tasks. For instance, in certain simulators that allow the construction of cities it is required to navigate the world with the directional pad while toggling editing modes to select, rotate and adjust the building to be placed. In contrast, tangible interfaces may seamlessly handle such processes easily as they account for mappings to several inputs and therefore support manipulations effectively.

Thus, input controllers can clearly be a barrier for creativity when it is not easy and comfortable handling the system to do exactly what users want, leading them to disengagement and leave the activity, impacting negatively on user (learning) performance. Such tendency towards tangibility has however some issues that may make it less interesting in terms of mass market, investment and acquisition of underused devices. Although tangibles normally entail more specific controllers, as soon as newer natural interaction technologies (e.g. motion and depth-sense trackers like Kinect) are successfully combined with new general and affordable tangibles, contributions to creation interfaces will become more relevant.

4.4 Measurement and Assessment

Measurement and assessment are complex and they are challenges more related to pedagogical issues. Defining creativity accurately is sometimes difficult, as mentioned above, and consequently suitable measures are also hard to be defined and operationalized because they depend on the task context. Although there are assessment models [28], the needed measures are not easy to obtain, and it therefore hinders the collection of clear evidences that eventually prove the degree in which learning environments bring benefits and advantages in developing creativity, what would give value and encouragement to adopt them. This lack hampers the evolution and the introduction of structural changes in the educational system so that instructionist activities become replaced by other creativity-related activities.

This reality requires of multidisciplinary joint work aiming at the integration of measurement instruments from the pedagogy and psychology into computer-mediated activities. The relationship between interactions (between both the machine and peers) and creativity indicators must be studied in order to devise computational proposals that automatically detect creativity cues to be used not only for assessment but also to facilitate the necessary scaffolding or the construction of user profiles that could also enhance other context based learning experiences.

4.5 Storytelling and Coordination Schemes

A highly creative activity is that of inventing and telling stories. Such activity normally also considers the creation of the characters and props which can be used during the performance, developing more artistic and handcraft aspects.

Computational environments oriented to this kind of activity are usually based on either rules or timeline models involving virtual entities in order to make the story progress. These environments must evolve towards open or live performances, in which available smart elements enrich the performances and unleash creative situations. Hence, it is interesting to come up with new forms of storytelling that includes smart technologies so that a mixture of live and virtual performances can be used to express a story.

4.6 Group Activity Sensing

Proposals consider group activities to include discussion and reflection between participants. Interactions between individuals make it more difficult to analyze and discover how tasks are carried out. However, it would be useful being able to distinguish the relevant and meaningful actions among participants [29], since the environment would be able to determine the attitude, behavior, even the mood of participants as well as the state of the group during the progress of the activities.

4.7 Multi-device Integration

Interfaces can be seen as the creation tools that allow the manipulation of digital objects. Tools in a tool kit have primary functions that are addressed effectively. For instance, the screwdriver's primary function is either screw down or unscrew, and if it is used for any other function it will not work efficiently. Analogously, the interaction properties of each interface technology will determine their suitability to support one task or another. Hence, we must take advantage of the range of devices currently available (with different interfaces) which may offer different capabilities in terms of visualization and manipulation of digital objects. This can be understood as a sort of specialization in the use of available devices in order to take full advantage during interactions.

It brings the opportunity to consider integration of multiple devices, and therefore multi-device interaction frameworks should be devised, which allows sharing and exchange information among them seamlessly and easily [30], and which are used in situations supporting collaboration and co-operation effectively.

5 Conclusion

Related work showed that systems based on entertainment technology with attractive interfaces are being used for learning purposes. In those proposals the creation of artifacts is usually directed to programming simulations and the creation of entities. It is observed that proposals providing more creative capabilities are usually based on WIMP interfaces, therefore negatively impacting on the role of collaboration and active participation.

Covering these desirable characteristics, this paper has summarized the main findings of a new approach developed within the scope of the CreateWorlds project,

which relies on an interactive tabletop interface in order to foster discussion, action and reflection when supporting the creation of digital game ecosystems. Finally, several future challenges have been enunciated that should be explored in order to technologically move towards the construction of creative ambient intelligent game-oriented learning environments.

Acknowledgements

This work received financial support from the Spanish Ministry of Education under the National Strategic Program of Research and Project TIN2010-20488. Our thanks to Polimedia/UPV for the support in computer hardware.
This work is also supported by a postdoctoral fellowship within the VALi+d program from Conselleria d'Educació, Cultura i Esport (Generalitat Valenciana) to A. Catalá (APOSTD/2013/013).

References

1. European Commission: Progress Report: Chapter IV - Enhancing creativity and Innovation, including entrepreneurship at all levels of education and training. In Progress Towards the Lisbon objectives in education and training, indicators and benchmarks (2009)
2. Aleinikov, A., Kackmeister, S., Koenig R. (Eds.): Creating creativity: 101 definitions. Midland, MI: Alden B. Dow Creativity Center, Northwoods University (2000)
3. Sawyer, R. K.: Explaining creativity: The science of human innovation. New York: Oxford University Press (2006)
4. Maloney, J. et al.: Scratch: A Sneak Preview. In: International Conference on Creating, Connecting and Collaborating through Computing, Kyoto, Japan, pp. 104-109 (2004)
5. Fernaeus, Y., Tholander, J.: Finding design qualities in a tangible programming space. In: SIGCHI conference on Human Factors in computing systems, ACM, pp. 447-456 (2006)
6. Parkes, A.J. et al.: Topobo in the wild: longitudinal evaluations of educators appropriating a tangible interface. In: SIGCHI conference on Human factors in computing systems, Florence, Italy, pp. 1129-1138 (2008)
7. Michael, D., Chen, S.: Serious Games: Games that Educate, Train, and Inform. Course Technology PTR, Mason, USA (2005)
8. Papert, S: Different Visions of Logo, Computers in the Schools, 2(2-3) (1985)
9. Suzuki, H., Kato, H.: Interaction-level support for collaborative learning: AlgoBlock-an open programming language. In: The first international conference on Computer support for collaborative learning. John L. Schnase and Edward L. Cunnius (Eds.). L. Erlbaum Associates Inc., Hillsdale, NJ, USA, pp. 349-355 (1995)
10. Cockburn, A., Bryant, A.: Cleogo: Collaborative and Multi-Metaphor Programming for Kids. In: the Third Asian Pacific Computer and Human Interaction, pp.189-194 (1998)
11. Gallardo, D., Fernàndez-Julià, C., Jordà, S.: TurTan: A Tangible programming language for creative exploration. In: the 2008 IEEE international workshop on horizontal interactive human-computer-systems, Amsterdam, The Netherlands (2008)
12. Horn, M.S., Jacob, R.J.K.: Designing Tangible Programming Languages for Classroom Use. In: the 1st international conference on tangible and embedded interaction, pp. 159-162 (2007)

13. Leitner, J. et. al.: Physical interfaces for tabletop games. ACM Comput. Entertain. 7, 4, Article 61, December (2009)
14. Kelleher, C., Pausch, R.: Using storytelling to motivate programming. Magazine Communications of the ACM - Creating a science of games CACM 50:7, 58-64 (2007)
15. Begel, A.: LogoBlocks: A Graphical Programming Language for Interacting with the World. MIT (1996)
16. Repenning A., Ioannidou A., Zola J.: AgentSheets: End-User Programmable Simulations. Journal of Artificial Societies and Social Simulation, 3:3 (2000)
17. Lu, F. et. al.: ShadowStory: Creative and Collaborative Digital Storytelling Inspired by Cultural Heritage. In: SIGCHI conference on Human factors in computing systems, pp. 1919-1928 (2011)
18. Ellis H., Heppell S., Kirriemuir J., Krotoski A., McFarlane A.: Unlimited Learning: The role of computer and video games in the learning landscape. ELSPA: Entertainment and Leisure Software Publishers Association (2006)
19. Abt, C.: Serious Games. Viking Press, New York, USA (1970)
20. Dillenbourg, P., Evans, M.: Interactive tabletops in education. International Journal of Computer-Supported Collaborative Learning, 6:4, 491-514 (2011)
21. Catala, A., Garcia-Sanjuan, F., Pons, P., Jaen, J., Mocholi, J.A. AGORAS: Towards collaborative game-based learning experiences on surfaces. In: Cognition and Exploratory Learning in Digital Age, IADIS, pp. 147-154 (2012)
22. Catala, A., Pons, P., Jaen, J., Mocholi, J.A., Navarro, A.: A meta-model for dataflow-based rules in smart environments: Evaluating user comprehension and performance, Science of Computer Programming, In press, Elsevier, available online 20 July 2012
23. Pons, P., Catala, A., Jaen, J.: TanRule: A Rule Editor for Behavior Specification on Tabletops. In: Extended Abstracts of the ACM Tangible, Embedded and Embodied Interaction TEI (2013).
24. Catala, A. et al.: Exploring Direct Communication and Manipulation on Interactive Surfaces to Foster Novelty in a Creative Learning Environment. In: International Journal of Computer Science Research and Application, 2:1, 15-24 (2012)
25. Catala, A., Jaen, J., Martinez, A. A., Mocholi, J.A.: AGORAS: Exploring Creative Learning on Tangible User Interfaces. In: the IEEE 35th Annual Computer Software and Applications Conference, pp.326-335 (2011)
26. Catala, A., Jaen, J., van Dijk, B., Jordà, J.: Exploring tabletops as an effective tool to foster creativity traits. In: the Sixth ACM International Conference on Tangible, Embedded and Embodied Interaction-TEI, pp. 143-150, Kingston, Canada, (2012)
27. Catala, A., Garcia-Sanjuan, F., Jaen, J., Mocholi, J.A.: TangiWheel: A Widget for Manipulating Collections on Tabletop Displays Supporting Hybrid Input Modality. J. Comput. Sci. Technol., (27)4:811-829 (2012)
28. Ellis, S., Barrs, M.: The assessment of creative learning in Creative Learning. Creative Partnerships, Arts Council England, 14 Great Peter Street, London, SW1P 3NQ (2008)
29. Vinciarelli, A., Pantic, M., Bourlard, H., Pentland, A.: Social signal processing: state-of-the-art and future perspectives of an emerging domain. In: the 16th ACM international conference on Multimedia, ACM, pp. 1061-1070, New York, NY, USA, (2008)
30. Marquardt, N., Hinckley, K., Greenberg, S.: Cross-device interaction via micro-mobility and f-formations. In: the 25th annual ACM symposium on User interface software and technology, pp. 13-22, New York, NY, USA: ACM (2012)

Evolving Aesthetic Maps for a Real Time Strategy Game

Raúl Lara-Cabrera, Carlos Cotta and Antonio J. Fernández-Leiva

Department "Lenguajes y Ciencias de la Computación", ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 Málaga – Spain
{raul,ccottap,afdez}@lcc.uma.es

Abstract. This paper presents a procedural content generator method that have been able to generate aesthetic maps for a real-time strategy game. The maps has been characterized based on several of their properties in order to define a similarity function between scenarios. This function has guided a multi-objective evolution strategy during the process of generating and evolving scenarios that are similar to other aesthetic maps while being different to a set of non-aesthetic scenarios. The solutions have been checked using a support-vector machine classifier and a self-organizing map obtaining successful results (generated maps have been classified as aesthetic maps).

Keywords: Procedural content generation, game aesthetics, computational intelligence, real-time strategy games

1 Introduction

The video-game industry has become one of the most important component in the entertainment business, with a total consumer spent of 24.75 billion US dollars in 2011 [5]. The quality and appealing of video-games used to rely on their graphical quality until the last decade, but now, their attractiveness has fallen on additional features such as the music, the player immersion into the game and interesting story-lines. It is hard to evaluate how much amusing a game is because this evaluation depends on each player, nevertheless there is a relationship between player satisfaction and fun.

Procedural Content Generation (PCG) [12] includes algorithms and techniques dedicated to produce game content automatically, providing several advantages to game developers, such as reduced memory consumption, the possibility of create endless video-games (i.e. the game changes every time a new game is started) and a reduction in the expense of creating the game content. These benefits are well known by the industry as demonstrated by the use of PCG techniques during the development of commercial games such as *Borderlands*, *Borderlands 2*, *Minecraft* and *Spore*.

Search-based procedural content generation [22] (SBPCG) techniques apply a generate and test scheme, that is, the content is firstly generated and then

evaluated according to some criteria. This evaluation sets the quality level of the generated content automatically. Then, new content is created based on previous evaluations. This process, which should be automated, is repeated until the content reach a certain quality level, hence making evolutionary algorithms a perfect tool for this process. In this work we have used a SBPCG technique to generate aesthetic maps for a real-time strategy (RTS) game. This method is based on a multi-objective evolution strategy that generate maps which are similar to other aesthetic maps and different from other non-aesthetic maps at the same time.

2 Background

Real-time strategy games offer a large variety of fundamental AI research problems [1] such as adversarial real-time planning, decision making under uncertainty, opponent modelling, spatial and temporal reasoning and resource management. This genre of game has been widely used as a test-bed for AI techniques [14]. *Planet Wars* is a real-time strategy game based on *Galcon* and used in the *Google AI Challenge 2010*. The objective is to conquer all the planets on the map or eliminate all the opponents. Games take place on a map on which several planets are scattered. These planets are able to host ships and they can be controlled by any player or remain neutral if no player conquer them. Moreover, planets have different sizes, a property that defines their growth rate (i.e., the number of new ships created every time step, as long as the planet belongs to some player). Players send fleets of ships from controlled planets to other ones. If the player owns the target planet the number of fleet's ships is added to the number of ships on that planet, otherwise a battle takes place in the target planet: ships of both sides destroy each other so the player with the highest number of ships owns the planet (with a number of ships determined by the difference between the initial number of ships). The distance between the planets affects the required time for a fleet to arrive to her destination, which is fixed during the flight (i.e., it is not possible to redirect a fleet while it is flying).

PCG for *Planet Wars* involves in this case generating the maps on which the game takes place. The particular structure of these maps can lead to games exhibiting specific features. In previous work [15,16] we focused on achieving balanced (i.e., games in which none of the players strongly dominates her opponent) and dynamic (i.e., action-packed) games. PCG techniques are usually employed to generate maps, as exhibited by the large number of papers on this topic [12]. For example, Mahlmann et al. [18] presented a search-based map generator for an simplified version of the RTS game *Dune*, which is based on the transformation of low resolution matrices into higher resolution maps by means of cellular automata. Frade et al. introduced the use of genetic programming to evolve maps for videogames (namely terrain programming), using either subjective human-based feedback [9,10] or automated quality measures such as accessibility [8] or edge-length [11]. Togelius et al. [21] designed a PCG system

capable of generating tracks for a simple racing game from a parameter vector using a deterministic genotype-to-phenotype mapping.

Maps are not the only content that is generated with these methods. For example, Font et al. [6] presented initial research regarding a system capable of generating novel card games. Collins [2] made an introduction to procedural music in video games, examining different approaches to procedural composition and the control logics that have been used in the past. The authors of [19] have created a prototype of a tool that automatically produce design pattern specications for missions and quest for the game Neverwinter Nights.

3 Materials and methods

As stated before, this work focuses on the design of aesthetic maps for the RTS game *Planet Wars*. This section describes the mechanisms that helped us to achieve our goal: firstly, there is a description about how the maps have been represented and characterized in order to get better aesthetics; next, there is a detailed explanation of the evolutionary algorithm used to generate the maps.

3.1 Representation and characterization

Game's maps are sets with a certain number of planets n_p located on a 2D plane. These planets are defined by their position on the plane (coordinates (x_i, y_i)), their size s_i and a number of ships w_i. The size s_i defines the rate at which a planet will produce new ships every turn (as long as the planet is controlled by any player) while the remaining parameter, w_i, indicates the number of ships that are defending that planet. Hence, we can denote a map as a list $[\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, \cdots, \boldsymbol{\rho}_{n_p}]$, where each $\boldsymbol{\rho}_i$ is a tuple $\langle x_i, y_i, s_i, w_i \rangle$. A playable map needs to specify the initial home planets of the players, which have been fixed as the first two planets $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$ because of simplicity. The number of planets n_p is not fixed and should range between 15 and 30 as specified by the rules of the *Google AI Challenge 2010*. This variable number of planets makes part of the self-adaptive evolutionary approach described later on.

In order to evaluate the generated maps' aesthetics , we have defined several measurements that characterize them. These are indicators related to the spatial distribution of the planets and their features, such as size and number of ships:

– Planet's geometric distribution: Let $\boldsymbol{p}_i = (x_i, y_i)$ be the coordinates of the i-th planet and N the total number of planets, so we defined the average distance between planets μ_d and the standard deviation of these distances σ_d as follows:

$$\mu_d = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \|\boldsymbol{p}_i - \boldsymbol{p}_j\| \tag{1}$$

$$\sigma_d = \sqrt{\frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\|\boldsymbol{p}_i - \boldsymbol{p}_j\| - \mu_d)^2} \tag{2}$$

– Planet's features: Let s_i and w_i be the size (i.e. growth rate) and number of ships, respectively, of the i-th planet, then we specified the average and standard deviation of these sizes (μ_s and σ_s respectively) and the Pearson's correlation between the planet's size and the number of ships on it ρ as follows:

$$\mu_s = \frac{1}{N}\sum_{i=1}^{N} s_i \tag{3}$$

$$\sigma_s = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(s_i - \mu_s)^2} \tag{4}$$

$$\rho = \frac{\sum_{i=1}^{N} s_i w_i - N\mu_s\mu_w}{N\sigma_s\sigma_w} \tag{5}$$

where μ_w and σ_w are the average and standard deviation of ships, respectively.

These measures has been applied to compare the likelihood between maps in the following way: each map is characterized by a tuple $\langle \mu_d, \sigma_d, \mu_s, \sigma_s, \rho \rangle$, then the euclidean distance between these tuples defined the similarity among the planets they represented. Additionally, we specified two sets of maps, one of them containing 10 maps with good aesthetics and the other one including 10 non-aesthetic maps. These sets made up a baseline to compare with in a way that the goal of generating aesthetic maps turned into an optimization problem about minimizing the distance between generated and aesthetics maps while maximizing their distance to non-aesthetic maps. The latter was necessary to insert diversity into the set of generated maps in order to avoid the generation of maps that were very similar to the aesthetic ones.

3.2 Evolutionary map generation

This procedural map generator used a multi-objective self-adaptive $(\mu + \lambda)$ evolution strategy (with $\mu = 10$ and $\lambda = 100$) whose objectives were to reduce the distance between the generated maps and those considered aesthetics and to increase the distance to non-aesthetic maps, in order to obtain procedurally generated aesthetic maps. Mixed real-integer vectors represented the solutions (i.e., planets): planet's coordinates (x_i, y_i) are real-valued numbers but sizes s_i and initial number of ships w_i are positive integers.

Due to this situation we have considered a hybrid mutation operator that performed different methods for parameters of either type: for real-valued parameters, it used a Gaussian mutation; as for integer variables, it considered a method that generates suitable integer mutations [17,20] – see also [16]. The latter is similar to the mutation of real values but it uses the difference of two geometrically distributed random variables to produce the perturbation instead of the normal distributed random variables used by the former. In either case, the

parameters that ruled the mutation were also a part of the solutions, thus providing the means for self-adapting them. More precisely, regarding real-valued parameters $\langle r_1, ..., r_n \rangle$ they are extended with n step sizes, one for each parameter, resulting in $\langle r_1, ..., r_n, \sigma_1, ..., \sigma_n \rangle$. The mutation method is specified as follows:

$$\sigma_i' = \sigma_i \cdot e^{\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)} \tag{6}$$

$$r_i' = r_i + \sigma_i \cdot N_i(0,1) \tag{7}$$

where $\tau' \propto 1/\sqrt{2n}$, and $\tau \propto 1/\sqrt{2\sqrt{n}}$. A boundary rule is applied to step-sizes to forbid standard deviations very close to zero: $\sigma_i' < \epsilon_0 \Rightarrow \sigma_i' = \epsilon_0$ (in this algorithm, σ_0 comprises a 1% of the parameter's range). In the case of integer-valued parameters $\langle z_1, ..., z_m \rangle$ they are extended in a similar way as are real-valued parameters, resulting in $\langle z_1, ..., z_m, \varsigma_1, ..., \varsigma_m \rangle$. The following equations define the mutation mechanism:

$$\varsigma_i' = \max(1, \varsigma_i \cdot e^{\tau \cdot N(0,1) + \tau' \cdot N(0,1)}) \tag{8}$$

$$\psi_i = 1 - (\varsigma_i'/m) \left(1 + \sqrt{1 + \left(\frac{\varsigma_i'}{m} \right)^2} \right)^{-1} \tag{9}$$

$$z_i' = z_i + \left\lfloor \frac{ln(1 - U(0,1))}{ln(1 - \psi_i)} \right\rfloor - \left\lfloor \frac{ln(1 - U(0,1))}{ln(1 - \psi_i)} \right\rfloor \tag{10}$$

where $\tau = 1/\sqrt{2m}$ and $\tau' = 1/\sqrt{2\sqrt{m}}$.

We considered a "cut and splice" recombination operator that recombines two individuals by swapping cut pieces with different sizes. This operator selects one cut point for each individual and then exchanges these pieces, getting two new individuals with a different number of planets in relation to their parents. This endows the algorithm with further self-adaptation capacities, hence affecting the complexity of the maps, i.e., the number of planets in the solutions.

As described in section 3.1, we characterized every map as a vector of five elements so the euclidean distance between these vectors measures the likelihood between them, hence the fitness function used to evaluate the individuals is, precisely, the median euclidean distance from the individual to every map from the set of aesthetics (minimization objective) and non-aesthetics (maximization objective) maps.

Considering that we had a multi-objective optimization problem, we decided to use the selection method of the Non-dominated Sorting Genetic Algorithm II (*NSGA-II*) [4].

4 Experimental Results

We have used the DEAP library [7] to implement the aforementioned algorithm. We have run 20 executions of the algorithm during 100 generations each. We have also computed the cumulative non-dominated set of solutions from every

execution – see figure 1. As we can see, there is a linear relationship between both distances in the middle range of the front. This hints at the density of the search space and the feasibility of linearly trading increasing distance to good maps by increasing distance to bad maps.

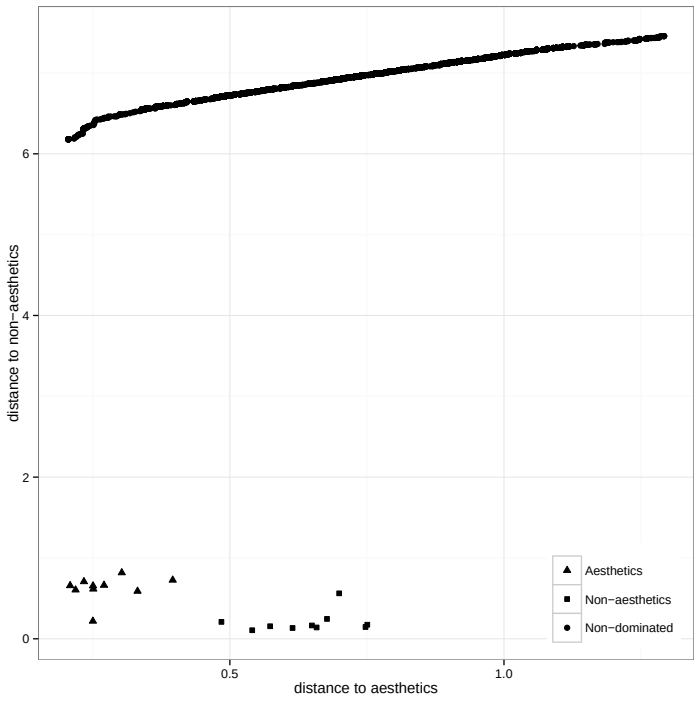


Fig. 1. Cumulative set of non-dominated generated solutions (circle) and maps from aesthetic (triangle) and non-aesthetic (square) baseline sets.

Figure 2 shows how are distributed the values of the different variables that make up the characterization vector of a map. Note that there are some variables that are similar in both aesthetics and non-aesthetic maps, such as σ_d and σ_s. However, this variable is higher in the case of the non-dominated maps, which should explain the high distance between many solutions in the front and the baseline maps, as seen in figure 1. Another interesting observation is the highly distributed values of μ_d in the non-dominated maps, which probably means that this variable has an uncertain effect over the fitness and hence the search space for this variable is wider with respect to other variables.

In order to check the validity of the generated maps, we built a support vector machine [3] (SVM) to classify maps (namely, characterization vectors) as aesthetic and non-aesthetic. Support vector machines are supervised learning

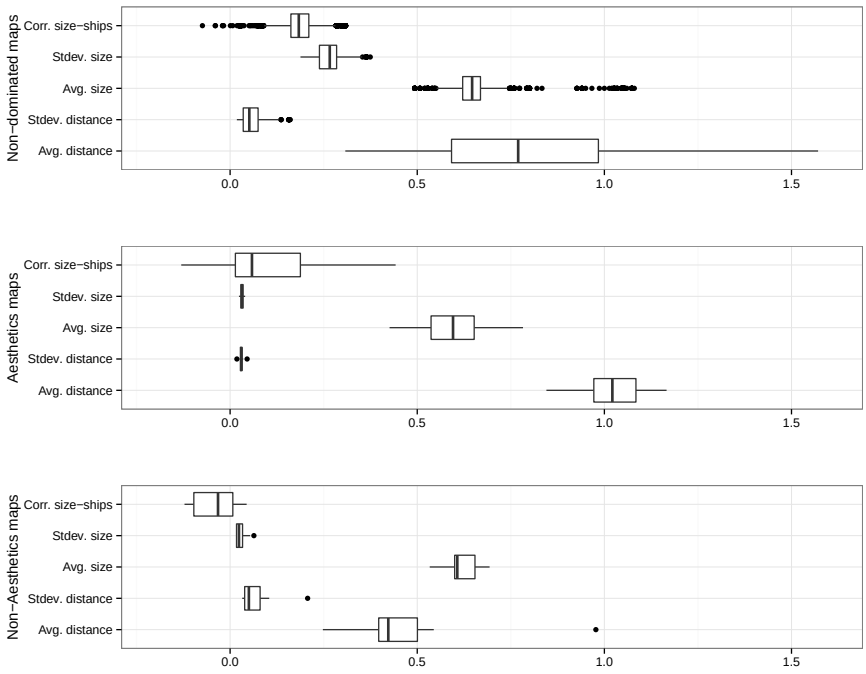


Fig. 2. Characterization variables for both non-dominated and baseline maps

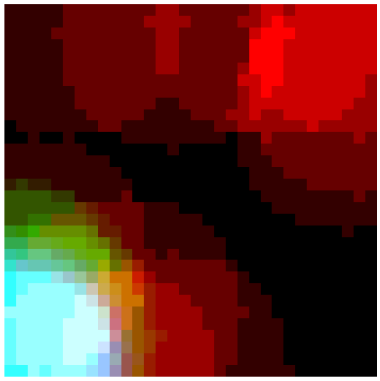


Fig. 3. Map's distribution over the SOM. Red for non-aesthetic, green for aesthetic and blue for non-dominated.

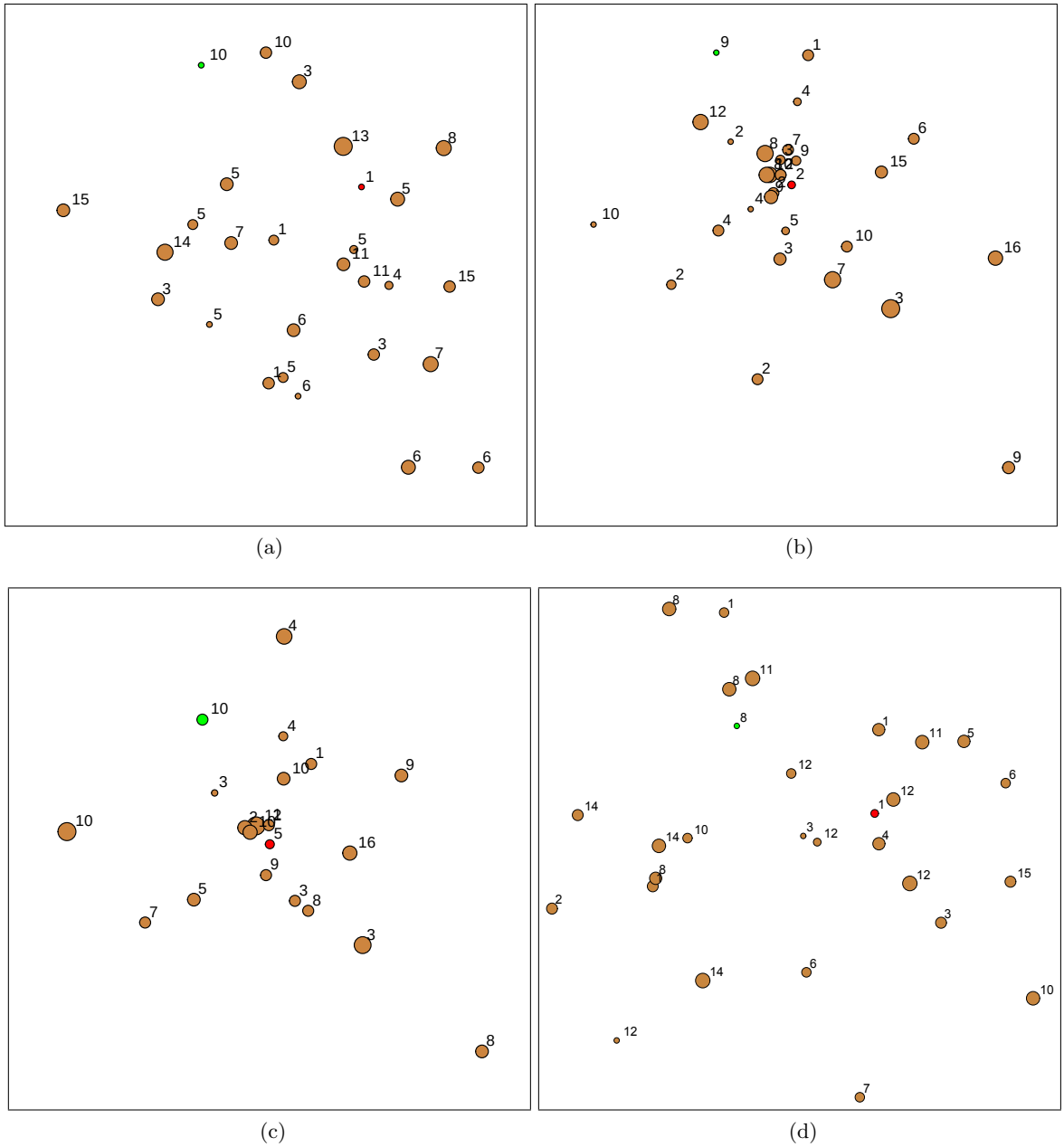


Fig. 4. Examples of generated maps

models that recognize patterns and analyze data. They are able to perform linear and non-linear classification. The SVM was trained using the same sets of maps that the evolutionary algorithm has used to calculate the fitness. This SVM classified as aesthetic every map out of the 4289 non-dominated maps, which led us to think that the algorithm is capable of generating aesthetic maps.

In addition to the aforementioned classifier, we created a self-organizing map (SOM) [13] with 32×32 process units over a non-toroidal rectangular layout, using the same maps as the training set. Self-organizing maps are artificial neural networks that are trained using unsupervised learning to generate a discretized representation of the input space. As we can see in figure 3, this SOM established a separation between non-aesthetic (red zones, upper-right) and aesthetic maps (green zones, lower-left). Moreover, generated maps (blue zones) shared the same region as aesthetic maps, hence they should be considered aesthetic as well.

5 Conclusions

We have performed an initial approach towards the procedural aesthetic map generation for the RTS game *Planet Wars*. We have defined a method of map characterization based on several of its maps' geometric and morphologic properties in order to evaluate how aesthetic a map is. We have used two sets of maps (aesthetics and non-aesthetics) as a baseline to compare with, and an evolution strategy whose objectives are minimize and maximize the distance of the generated maps to the aesthetics and non-aesthetics maps of the baseline. The solutions have been tested with a SVM and a SOM. The SVM has classified each solution as aesthetic while the SOM was able to make a separation between aesthetic and non-aesthetic maps (the generated maps shared the same region as the aesthetic maps).

We have used a geometric characterization of the maps (namely planets' coordinates), which means that this characterization is affected by rotation, translation and scaling, thus suggesting the use of other kind of measurements, such as topological variables, as a potential line of future research.

Acknowledgments This work is partially supported by Spanish MICINN under project ANYSELF[1] (TIN2011-28627-C04-01), and by Junta de Andalucía under project P10-TIC-6083 (DNEMESIS[2]).

References

1. Buro, M.: RTS games and real-time AI research. In: Behavior Representation in Modeling and Simulation Conference. vol. 1. Curran Associates, Inc. (2004)
2. Collins, K.: An introduction to procedural music in video games. Contemporary Music Review 28(1), 5–15 (2009)

[1] `http://anyself.wordpress.com/`
[2] `http://dnemesis.lcc.uma.es/wordpress/`

3. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (1995)
4. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Schoenauer, M., et al. (eds.) Parallel Problem Solving from Nature VI. Lecture Notes in Computer Science, vol. 1917, pp. 849–858. Springer-Verlag, Berlin Heidelberg (2000)
5. Entertainment Software Association: Essential facts about the computer and video game industry (2012), http://www.theesa.com/facts/pdfs/esa_ef_2012.pdf
6. Font, J., Mahlmann, T., Manrique, D., Togelius, J.: A card game description language. In: Esparcia-Alczar, A. (ed.) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 7835, pp. 254–263. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-37192-9_26
7. Fortin, F.A., Rainville, F.M.D., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research 13, 2171–2175 (jul 2012)
8. Frade, M., de Vega, F., Cotta, C.: Evolution of artificial terrains for video games based on accessibility. In: Di Chio, C., et al. (eds.) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 6024, pp. 90–99. Springer-Verlag, Berlin Heidelberg (2010)
9. Frade, M., de Vega, F.F., Cotta, C.: Modelling video games' landscapes by means of genetic terrain programming - a new approach for improving users' experience. In: Giacobini, M., et al. (eds.) Applications of Evolutionary Computing. Lecture Notes in Computer Science, vol. 4974, pp. 485–490. Springer-Verlag, Berlin Heidelberg (2008)
10. Frade, M., de Vega, F.F., Cotta, C.: Breeding terrains with genetic terrain programming: The evolution of terrain generators. International Journal of Computer Games Technology 2009 (2009)
11. Frade, M., de Vega, F.F., Cotta, C.: Evolution of artificial terrains for video games based on obstacles edge length. In: IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE (2010)
12. Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A.: Procedural content generation for games: A survey. ACM Trans. Multimedia Comput. Commun. Appl. 9(1), 1:1–1:22 (Feb 2013), http://doi.acm.org/10.1145/2422956.2422957
13. Kohonen, T.: The self-organizing map. Proceedings of the IEEE 78(9), 1464–1480 (1990)
14. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: A Review of Computational Intelligence in RTS Games. In: Ojeda, M., Cotta, C., Franco, L. (eds.) 2013 IEEE Symposium on Foundations of Computational Intelligence. pp. 114–121
15. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: Procedural map generation for a RTS game. In: Leiva, A.F., et al. (eds.) 13th International GAME-ON Conference on Intelligent Games and Simulation, pp. 53–58. Eurosis, Malaga (Spain) (2012)
16. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: A procedural balanced map generator with self-adaptive complexity for the real-time strategy game planet wars. In: Esparcia-Alcázar, A., et al. (eds.) Applications of Evolutionary Computation, pp. 274–283. Springer-Verlag, Berlin Heidelberg (2013)
17. Li, R.: Mixed-integer evolution strategies for parameter optimization and their applications to medical image analysis. Ph.D. thesis, Leiden University (2009)
18. Mahlmann, T., Togelius, J., Yannakakis, G.N.: Spicing up map generation. In: Chio, C.D., et al. (eds.) Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol. 7248, pp. 224–233. Springer-Verlag, Málaga, Spain (2012)

19. Onuczko, C., Szafron, D., Schaeffer, J., Cutumisu, M., Siegel, J., Waugh, K., Schumacher, A.: Automatic story generation for computer role-playing games. In: AIIDE. pp. 147–148 (2006)
20. Rudolph, G.: An evolutionary algorithm for integer programming. In: Davidor, Y., Schwefel, H.P., Männer, R. (eds.) Parallel Problem Solving from Nature III, Lecture Notes in Computer Science, vol. 866, pp. 139–148. Springer-Verlag, Jerusalem, Israel (1994)
21. Togelius, J., De Nardi, R., Lucas, S.: Towards automatic personalised content creation for racing games. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on. pp. 252–259 (2007)
22. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games 3(3), 172–186 (2011)

Code Reimagined: Gamificación a través de la visualización de código.

J.J. Asensio, A.M. Mora,
P. García-Sánchez, J.J. Merelo

Departamento de Arquitectura y Tecnología de Computadores.
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación.
Universidad de Granada, España
{asensio, amorag, pablogarcia, jmerelo}@ugr.es

Resumen Las herramientas de visualización de software juegan un papel muy importante en el desarrollo y mantenimiento de sistemas de software complejos. Del mismo modo, también son útiles durante el aprendizaje de los conceptos básicos de programación, los cuales son aprendidos o enseñados a edades cada vez más tempranas. En esta línea, las técnicas de gamificación que intentan hacer más atractivas las tareas de programación juegan un papel importante. Para su aplicación, es necesario establecer formas innovadoras de representación que habiliten el uso de metáforas conceptuales llamativas para el estudiante. En este trabajo se propone una herramienta de visualización de código a nivel de método, basada en mapa, utilizando videojuegos de plataformas como metáfora, en este caso, una versión libre de Super Mario Bros.

1. Introducción

La visualización de software consiste en representar gráficamente (en 2D o 3D) de forma estática o dinámica (con animaciones) información relacionada con la estructura, tamaño, historia o comportamiento de un sistema software. El objetivo es facilitar la comprensión de diferentes aspectos del sistema. Esta visualización permite documentar complejos sistemas y detectar anomalías, siendo también especialmente útil durante el aprendizaje de los conceptos.

Para representar la estructura del software (generalmente orientado a objetos), normalmente, el enfoque consiste en representar las dependencias existentes entre las diferentes entidades. Estas dependencias puede ser modeladas como un grafo por lo que la visualización de grafos resulta adecuada. El software a un nivel más básico, es decir, a nivel de método, puede ser también representado de esta forma, mediante un diagrama de flujo. Este diagrama está basado en nodos y enlaces.

Sin embargo, el código de un método cumple además con las reglas sintácticas del lenguaje, por lo que también puede ser representado mediante su árbol sintáctico abstracto. En este árbol los nodos representan los diferentes elementos del lenguaje utilizados y las aristas su relación de inclusión (por ejemplo, el nodo IF tiene como nodos hijos la parte THEN y la parte ELSE, cada una con

sus propios nodos hijos). A diferencia del diagrama de flujo, la visualización de un árbol es mucho más flexible. La jerarquía de un árbol puede ser visualizada mediante un mapa (sin utilizar flechas) [1].

Esta forma de visualización es susceptible de ser combinada con técnicas de gamificación [2]. Para ello, este trabajo propone una metáfora conceptual que consiste en utilizar el mapa de un videojuego de plataformas como representación del árbol sintáctico abstracto del programa. Esta visualización, además, permite la analogía dinámica entre ambos contextos semánticos, es decir, el protagonista del videojuego corre por el mapa y el ordenador ejecuta (corre) el programa. Esta metáfora es útil para el aprendizaje temprano de los elementos de control habituales en los lenguajes de programación.

Para la implementación de una herramienta de visualización dinámica de estas características se ha utilizado Java y Eclipse. El resto de este trabajo está organizado de la siguiente forma: en la sección 2 se explora el contexto taxonómico, se discute el concepto de legibilidad del código y su relación con algunas herramientas existentes para el aprendizaje temprano de la programación, se describen las técnicas gamificación y la importancia de su papel en la metáfora conceptual. En la sección 3 se define la herramienta propuesta concretando sus características. En la sección 4 se describen los detalles principales de la implementación. En la sección 5 se establecen las conclusiones obtenidas y la continuación de este trabajo junto con los agradecimientos.

2. Contexto

2.1. Contexto taxonómico del uso de gráficos en programación

Los sistemas que usan gráficos para programar, depurar y entender la computación en general han sido motivo de investigación desde el inicio de la informática. La idea de hacer más accesible la programación a cualquier tipo de usuario ha sugerido a la utilización de gráficos o lenguajes visuales para la confección de programas. De igual forma el uso de gráficos facilita la comprensión de lo que el programa hace así como su depuración cuando se enseña a programar a los estudiantes.

En este contexto muchas veces los términos utilizados para el uso de gráficos son a menudo informales o confusos. A finales de los 80 Myers [3] estableció una taxonomía más formal para los diferentes sistemas gráficos utilizados en aquel entonces y que continúa siendo válida. De esta forma establece la visualización de programas como el uso de gráficos que ilustran algún aspecto del programa o de su ejecución. En la figura 1 se muestra un ejemplo de visualización de código de forma estática.

En concreto se clasifican según se visualiza el código, los datos o el algoritmo. Tal visualización además puede ser estática o dinámica dependiendo de si existe algún tipo de animación mientras se ejecuta el programa [4]. La propuesta de este trabajo constituye una forma de visualización de código dinámica.

En este trabajo se propone una visualización dinámica de programas usando gamificación para mejorar la experiencia del programador, especialmente para

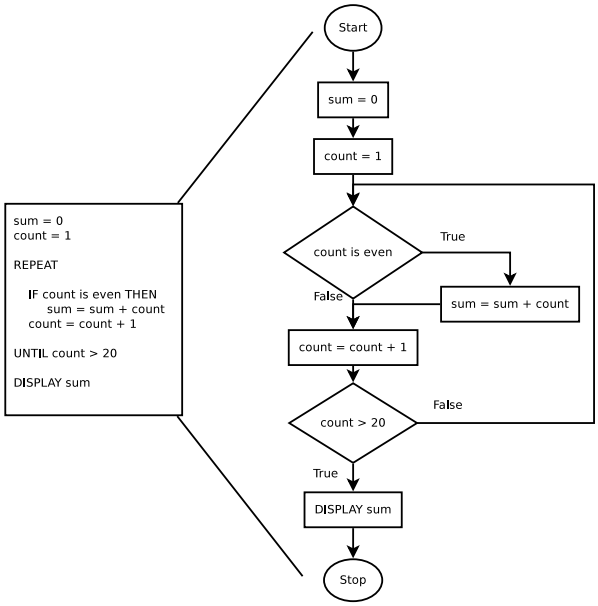


Figura 1. El diagrama de flujo visualiza el código asociado de forma estática.

el aprendiz, aunque no exclusivamente. El objetivo es añadir un valor expresivo, documentativo y estético al software a nivel de función o método.

La justificación se fundamenta en que el sistema visual humano está optimizado para procesar información multidimensional mientras que la programación convencional, sin embargo, es representada como un texto unidimensional. En este sentido, la capacidad del código de ser entendido por un humano, es decir, su legibilidad, es discutida a continuación y evaluada a través de algunas herramientas de ejemplo. En tales ejemplos se pone de manifiesto la necesidad de exploración y explotación de nuevas metáforas conceptuales que, aplicadas a un contexto puramente computacional, sean adecuadamente flexibles para expresar, resumir, documentar y comprender mejor el código. El videojuego de plataformas, como metáfora de programa, resulta ser especialmente adecuado para la descripción de los lenguajes Turing-completos, puesto que es universalmente conocido y ha surgido en el contexto de la programación.

2.2. Legibilidad

A menudo se hace analogía entre el texto de un programa y el texto narrativo. La legibilidad en este sentido, como concepto, es decir, la capacidad de que algo sea leído por un humano, es una idea general muy amplia. El término "leído" implica todas las fases del procesamiento del lenguaje a fin de que el significado sea entendido por el lector. Con esta definición en mente vemos claramente

diferencias sustanciales entre la legibilidad de un texto narrativo y el código de un programa.

La legibilidad del texto narrativo depende de la historia que se cuenta y de cómo el lector puede representar, mediante su imaginación, el significado natural de esta historia conforme a su experiencia. Si la historia es confusa o carece de sentido, el texto es poco legible. Sin embargo, es posible que la historia tenga pleno sentido y sea clara y que el texto siga siendo poco legible, por ejemplo, imaginemos un niño leyendo un artículo científico. En este caso la experiencia del lector es el factor predominante.

Si aplicamos esta misma idea al texto de un programa, enseguida nos damos cuenta de que la "historia" que expresa el programa poco tiene que ver con la experiencia del lector. Su significado existe en un contexto computacional, matemático-lógico completamente distinto, generalmente abstracto, y por tanto contrapuesto, al mundo real del lector. Para que el texto del programa sea legible por tanto sólo hay dos opciones:

- o bien forzamos el contexto semántico del programa para que se adecue a la experiencia natural del lector,
- o bien reforzamos la experiencia del lector en el contexto de la programación.

La semántica del mensaje. En el primer caso se intenta conseguir que el lector se introduzca en la programación partiendo de su experiencia natural. Su alcance, si bien importante, está limitado a la generación de programas cuya salida es multimedia y presenta alguna historia interactiva. El código usado para este fin concreto se limita a un conjunto específico de instrucciones. Incluso así, este código puede no ser legible hasta que finalmente se ejecute.

Algunas herramientas que utilizan este enfoque son:

- *Squeak Etoys*: Es un entorno de desarrollo dirigido a niños y un lenguaje de programación basado en prototipos y orientado a objetos [5]. Dirigido e inspirado por Alan Kay para promover y mejorar el constructivismo. Los principales influyentes en esta herramienta son Seymour Papert y el lenguaje Logo. La figura 2 muestra un programa de ejemplo.
- *Alice*: Alice [6] es un lenguaje de programación educativo libre y abierto orientado a objetos con un entorno de desarrollo integrado (IDE). Está programado en Java. Utiliza un entorno sencillo basado en arrastrar y soltar para crear animaciones mediante modelos 3D. Este software fue desarrollado por los investigadores de la Universidad Carnegie Mellon, entre los que destaca Randy Pausch.
- *Scratch*: Es un entorno de programación online [7] que facilita el aprendizaje autónomo. Fue desarrollado por el "the Lifelong Kindergarten group" en el Media Lab del MIT (Massachussets Institute of Tecnology) por un equipo dirigido por Mitchel Resnick y apareció por primera vez en el verano de 2007 [8]. La figura 3 muestra un programa de ejemplo. Los elementos de programación son representados como bloques de distinto color que encajan entre sí.

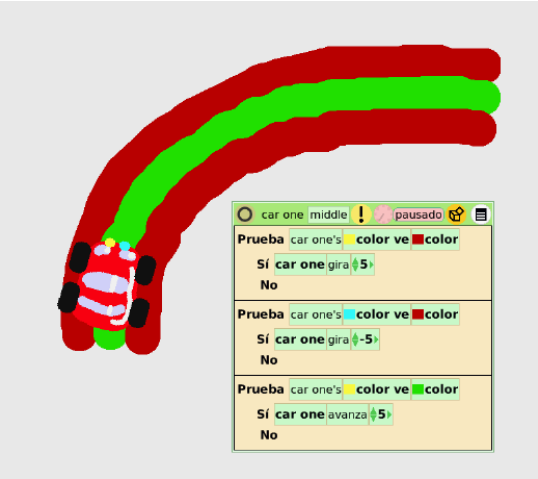


Figura 2. Un programa diseñado en Etoys para controlar un coche sigue-línea.

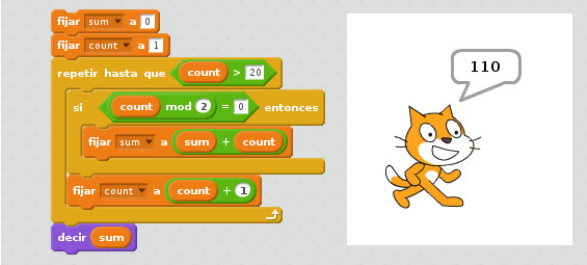


Figura 3. El mismo programa de la figura 1 diseñado con Scratch.

La experiencia del lector. El segundo caso, por otra parte, tiene un alcance mucho más amplio y por tanto, más difícil de conseguir. Es necesario trabajar multitud de conceptos como los algebraicos, lógicos y matemáticos desde una edad temprana en el colegio. Además, la naturaleza abstracta de estos conceptos dificulta su propia representación visual. En este sentido, algunos conceptos son más susceptibles que otros de ser visualizados, como por ejemplo, los conjuntos y sus operaciones, las funciones matemáticas, o los vectores, son más fáciles de visualizar que otros como la probabilidad, las derivadas o las diferentes partículas subatómicas. Esta visualización es importante durante el aprendizaje.

Como vemos, el concepto general de legibilidad en el ámbito de la programación no está claro. El motivo principal es claramente la confusión de no tener en cuenta que el verdadero lector de un programa no es una persona, sino una máquina.

Desde este enfoque más amplio, es posible arrojar más luz sobre la comprensión de los procesos subyacentes en el aprendizaje de la programación. De aquí la importancia de la documentación de código [9], a menudo descuidada. Es necesario abordar una automatización de generación de documentación legible entendiendo esta documentación, no sólo como un contenido añadido al programa, sino más bien como una característica intrínseca durante la generación del programa [10]. En este paradigma el programador escribe y tanto el humano como la máquina pueden leer. Puesto que la máquina y el ser humano leen de forma diferente es necesario proveer al programador de diferentes herramientas para generar paralelamente al código, distintas representaciones semánticamente representativas. Estas herramientas son especialmente importantes durante el aprendizaje.

En este trabajo se propone una representación metafórica para visualizar el flujo de un programa de forma dinámica. La semántica utilizada para esta metáfora toma prestada la experiencia de la persona al jugar a videojuegos para hacer más intuitivo y atractivo el proceso de aprendizaje en el uso de las estructuras de control básicas de un programa escrito en un lenguaje imperativo. La representación del código en forma de juego está motivada por la aplicación de técnicas de gamificación en el contexto educativo.

2.3. Gamificación

La gamificación consiste en el uso de mecánicas de juego o pensamiento de juego en un contexto ajeno al juego, con el fin de conseguir determinados objetivos. La gamificación aprovecha la predisposición psicológica del humano a participar en juegos. Las técnicas utilizadas consisten básicamente en tres tipos:

- *A)* Ofrecer recompensas por la realización de las tareas.
- *B)* Aprovechar la competitividad, haciendo visibles las recompensas entre los jugadores.
- *C)* Hacer más atractivas tareas ya existentes que normalmente pueden ser aburridas.

El campo de la educación presenta un gran potencial de crecimiento para la gamificación [11]. Existen multitud de ejemplos de uso de gamificación para el aprendizaje. Quizá el más llamativo es la escuela neoyorquina Quest to Learn [12], donde todo el proceso de aprendizaje está dirigido al juego.

Un ejemplo concreto de gamificación aplicado al aprendizaje de programación es Code Academy [13]. Esta plataforma online interactiva ofrece clases gratuitas de programación con diferentes lenguajes. La técnica de gamificación usada en este contexto es la de recompensar y motivar la competitividad de los estudiantes, es decir las técnicas A) y B).

En relación a la tercera técnica enumerada, entre muchos ejemplos, podemos destacar ParticleQuest. ParticleQuest trata de enseñar a los estudiantes de física las partículas subatómicas. La metáfora consiste en identificar las partículas subatómicas como personajes enemigos de un videojuego tipo RPG (Role-Playing Game). Las diferentes características de cada partícula identifican las habilidades y morfología de los diferentes personajes.

Figura 4. Ejemplo de gamificación: los enemigos en el juego ParticleQuest representan las diferentes partículas subatómicas.

La herramienta que se presenta en este trabajo se encuadra dentro de este tipo de técnicas. Pues al visualizar el texto correspondiente al código del programa como un escenario de un videojugo, resulta familiar y divertido al usuario, y se facilita así su comprensión e interpretación.

2.4. Metáfora conceptual

La idea de utilizar metáforas para la representación y comprensión de conceptos abstractos es fundamental durante el aprendizaje. En matemáticas se utilizan a menudo para enseñar todo tipo de conceptos. Algunos símbolos matemáticos, como por ejemplo, los paréntesis, pueden ser interpretados como metáforas, en el sentido de que encierran explícitamente un determinado contenido. Podríamos decir que su uso como símbolo posee un significado visual añadido. Existe sin embargo un inconveniente cuando se hace un uso intensivo de estos símbolos ya que pierden su capacidad explicativa, como se observa a continuación en el siguiente programa escrito en Lisp:

```
(defun factorial (n) (if (<= n 1) 1 (* n (factorial (- n 1)))))
```

Otro ejemplo es el uso de flechas para indicar transiciones de un punto a otro. Al igual que con los paréntesis, al aumentar el número de flechas, aumenta la dificultad de comprensión del diagrama. Además al ser utilizadas frecuentemente en diferentes contextos y con diferente significado, las flechas adquieren un cierto nivel de abstracción, que devalúa todavía más su capacidad aclaratoria.

Con estos ejemplos se intenta ilustrar la importancia y necesidad de encontrar representaciones visuales más concretas que faciliten la comprensión, documentación y el aprendizaje de forma diferenciada para los conceptos que se utilizan en programación y que además puedan ser generadas automáticamente a partir del código.

Para este propósito, es necesario conseguir un compromiso adecuado entre la abstracción asociada a los elementos visuales y su analogía semántica con el elemento representado. Además resulta también útil hacer que estas representaciones sean atractivas para el estudiante durante el aprendizaje.

3. Propuesta

Este trabajo propone una representación metafórica para visualizar dinámicamente los elementos del código de un programa. La metáfora utilizada está basada en técnicas de gamificación. La idea es hacer más atractiva la tarea de codificación en programación. Para ello se utilizan videojuegos de plataformas. Quizás el videjouego de este tipo más famoso es Super Mario, para este trabajo se ha utilizado una versión libre de este juego llamada Secret Maryo Chronicles. La metáfora utilizada consiste en visualizar un método o una función, como una plataforma en el escenario del videojuego. Todo lo que hay sobre esta plataforma será el contenido del método. El uso de las estructuras de control básicas de un programa se corresponden con otras plataformas o elementos del videojuego por los que el personaje principal se mueve. Esta analogía resulta muy intuitiva puesto que relaciona la dinámica de ambos contextos. En concreto, los elementos básicos que pueden ser representados incluyen las sentencias de control típicas como el condicional, el switch, los bucles y la instrucción de retorno como se explica a continuación:

- *Condicional*: Es representado mediante una plataforma elevada. Si la condición se cumple, el personaje saltará a la plataforma, si no, pasará por debajo.
- *Switch*: Es representado mediante varias plataformas elevadas apiladas en vertical una para cada caso. El personaje saltará a la plataforma del caso que se cumpla.
- *Bucle*: Es representado mediante una tubería o similar por donde el personaje se introduce y aparece al principio del bucle.
- *Retorno*: Una puerta representa la devolución de datos del método.

Para que el flujo de programa pueda ser representado al menos se requieren estos elementos. Existen multitud de videojuegos de plataformas que cumplen con estos requisitos. En este aspecto la visualización puede ser personalizada por el usuario eligiendo los gráficos relacionados con el juego que le resulte más atractivo. Además, otros elementos del lenguaje pueden ser representados mediante diferentes gráficos del juego. Por ejemplo, las expresiones pueden consistir en cajas que el personaje golpea para ver su contenido. Según este esquema el programa de las figuras 1 y 3 queda representado en la figura 5.

Otra característica interesante para dar más capacidad de expresión a la representación visual (en el sentido opuesto al ejemplo comentado de las flechas de la sección anterior), es la posibilidad de elegir diferentes texturas para las plataformas en el contexto del juego para diferentes partes del código. Por ejemplo, supongamos que hay una parte del código más interesante o que requiere

Figura 5. Visualización propuesta para el programa de ejemplo de la figura 1 y 3. El bucle está representado como la plataforma que hay entre los dos tubos, el IF dentro del bucle es otra plataforma interior y las diferentes expresiones son cajas.

mayor atención, entonces se pueden utilizar gráficos correspondientes a escenarios más difíciles del juego como mazmorras, escenarios de lava o nieve, que sean fácilmente identificables. En la figura 6 vemos un ejemplo.

Figura 6. Visualización donde se ha resaltado todo el código del bucle utilizando plataformas con nieve.

El lenguaje para el que se ha implementado la herramienta de visualización propuesta es Java. El motivo de esta elección es el amplio uso del mismo, y la facilidad que poseen las herramientas de desarrollo para ser extendidas.

4. Implementación

Para implementar esta herramienta de visualización dinámica se ha optado por utilizar el lenjuaje Java sobre Eclipse. Eclipse es una plataforma extensible para construir IDEs. Proporciona servicios básicos para utilizar varias herramientas que ayudan en las tareas de programación. Los desarrolladores de herramientas pueden contribuir a la plataforma envolviéndolas en forma de componentes llamados plugins. El mecanismo básico es añadir nuevos elementos de

procesamiento a los plugins existentes. Mediante un manifiesto XML se describe la forma en que el entorno de ejecución de Eclipse activará y manejará las instancias del plugin.

El plugin Code Reimagined consiste en una vista añadida a la perspectiva Java para mostrar la imagen generada a partir del código que se está editando.

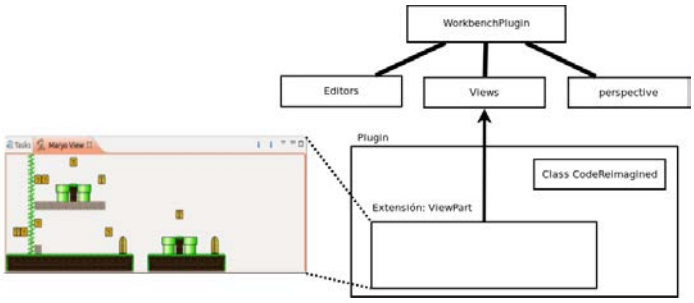


Figura 7. Esquema del plugin como una nueva vista en Eclipse.

Para analizar el código se ha usado el patrón "visitor" sobre el árbol sintáctico abstracto del código Java. Recorriendo este árbol se obtiene otro árbol cuyos nodos representan las áreas a dibujar en la vista. Estos nodos heredan de una clase abstracta y especifican cómo se pinta el elemento de programación dado y qué recuadro resulta de ello. Para obtener la imagen en la vista, basta hacer un recorrido de este árbol (sin importar el orden) y llamar al método que dibuja recursivamente los sprites de cada nodo.

Para representar la dinámica de Maryo moviéndose sobre la vista se ha implementado un "listener" para el cursor del editor de Java, de forma que el personaje aparecerá en el área correspondiente al elemento Java que haya inmediatemente después (figura 8). La manera de hacerlo ha sido utilizar una tabla hash indexando los nodos correspondientes a los elementos Java según su offset en el archivo.

Otra funcionalidad implementada es el posicionamiento del cursor en el editor Java, al hacer doble clic sobre el elemento correspondiente del mapa.

Este plugin se presentó al VII Concurso Universitario de Software Libre [1] quedando finalista en la final local de la Universidad de Granada. El código está disponible para descarga en `https://github.com/javiplay/code-reimagined`.

5. Conclusiones

En el contexto taxonómico de la visualización de programas y partiendo del concepto de legibilidad del código, este trabajo ha identificado la necesidad de

[1] `www.concursosoftwarelibre.org`

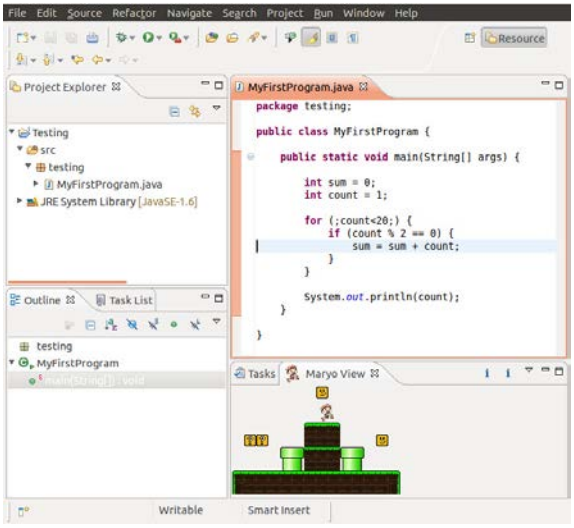


Figura 8. El plugin Code Reimagined muestra el programa de la figura 1 y 3 una vez integrado en el entorno Eclipse. Maryo aparece sobre el área dibujada correspondiente al elemento Java que hay a continuación a partir de la posición actual del cursor.

desarrollar técnicas de visualización automáticas, donde la metáfora conceptual juegue un papel básico, no sólo para el aprendizaje sino también para la documentación y la comprensión de las tareas y conceptos relacionados con la programación.

Para este tipo de visualización se ha establecido un principio básico de diseño que consiste en maximizar la precisión semántica de las representaciones minimizando la pérdida de analogía con los conceptos representados. Para la aplicación de este principio se ha comprobado la utilidad de las técnicas de gamificación, concretamente el uso de videojuegos de plataformas para representar el flujo de un programa.

La representación propuesta es independiente del lenguaje y por tanto, complementaria a cualquier software de programación y especialmente adecuado para herramientas de aprendizaje constructivista como Scratch. Una vez concretados los detalles de esta representación, se ha desarrollado una herramienta de visualización dinámica de código Java en forma de plugin para Eclipse.

Como líneas de actuación futura, además de añadir nuevas características, se propone la realización de experimentos para validar la efectividad de esta herramienta. En caso de obtener resultados positivos, se podría aplicar esta forma de visualización "gamificada", a otro tipo de diagramas relacionados con la programación orientada a objetos.

Agradecimientos

Este trabajo ha sido apoyado por el proyecto de excelencia EVORQ con referencia P08-TIC-03903 de la Junta de Andalucía, el proyecto Anyself con referencia TIN2011-28627-C04-02 del Ministerio de Innovación y Ciencia y el proyecto 83, CANUBE, concedido por el CEI-BioTIC de la Universidad de Granada.

Referencias

1. S, D.: Visualization basics. In: Software Visualization. Springer Berlin Heidelberg (2007) 15–33
2. Kumar, B.: Gamification in education-learn computer programming with fun. INTERNATIONAL JOURNAL OF COMPUTERS & DISTRIBUTED SYSTEMS **2**(1) (2012) 46–53
3. Myers, B.A.: Taxonomies of visual programming and program visualization. Journal of Visual Languages & Computing **1**(1) (1990) 97–123
4. Urquiza-Fuentes, J., Velázquez-Iturbide, J.Á.: A survey of successful evaluations of program visualization and algorithm animation systems. ACM Transactions on Computing Education (TOCE) **9**(2) (2009) 9
5. Etoys, S.: http://www.squeakland.org/ (2013)
6. Alice: http://www.alice.org/ (2013)
7. Scratch: http://scratch.mit.edu/ (2013)
8. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al.: Scratch: programming for all. Communications of the ACM **52**(11) (2009) 60–67
9. Tenny, T.: Program readability: Procedures versus comments. Software Engineering, IEEE Transactions on **14**(9) (1988) 1271–1279
10. Baecker, R.: Enhancing program readability and comprehensibility with tools for program visualization. In: Software Engineering, 1988., Proceedings of the 10th International Conference on, IEEE (1988) 356–366
11. Lee, J.J., Hammer, J.: Gamification in education: What, how, why bother? Academic Exchange Quarterly **15**(2) (2011) 146
12. Salen, K., Torres, R., Wolozin, L.: Quest to learn: Developing the school for digital kids. MIT Press (2011)
13. Codecademy: http://www.codecademy.com/ (2013)

Impacto de las nuevas tecnologías en la educación infantil

Fernando Palero, David Camacho

Departamento de Ingeniería Informática, Escuela Politécnica Superior,
Universidad Autónoma de Madrid,
C/Francisco Tomás y Valiente 11, 28049 Madrid.
fpalero86@gmail.com, david.camac@uam.es
http://aida.ii.uam.es

Resumen. Los colegios actualmente tienen la necesidad de nuevas metodologías de aprendizaje, estas metodologías están estrechamente relacionadas con el entorno socio-tecnológico actual. Aprovechando el auge actual de las nuevas tecnologías y el interés de los niños por aprender a utilizarlas, se han diseñado varios juegos educativos para niños comprendidos entre 3 y 5 años. Como elementos tecnológicos básicos se utilizará la pizarra digital y el sensor Kinect©. Estos dispositivos tienen la ventaja de permitir interactuar tanto a educadores como a niños con las manos desnudas, ayudando a que la interacción sea más intuitiva y favoreciendo el aprendizaje *kinestésico*. Con el método *kinestésico* la información que recibe el niño la asocia a las sensaciones y a los movimientos corporales. El presente trabajo muestra el diseño inicial de un conjunto de juegos destinados a niños de primeros cursos de educación infantil, así como una evaluación cualitativa inicial por parte de los educadores que han utilizado en varias sesiones los juegos desarrollados.

Palabras clave: pedagogía, educación infantil, psicomotricidad, aprendizaje innovador, aprendizaje visual, aprendizaje kinestésico.

1 Introducción

Actualmente con el auge de la tecnología, los niños y adolescentes se interesan por los dispositivos electrónicos, tabletas, Smartphone, videoconsolas, etc. Aprovechando la inquietud de los jóvenes y niños por la tecnología se han diseñado diversos videojuegos para niños comprendidos entre los 3 y 5 años. El objetivo de los juegos es ayudar a desarrollar las capacidades básicas de los niños adaptándose a los contenidos educativos del centro y que para el educador sea fácil cambiar estos contenidos en función del temario.

La integración de videojuegos educativos en el aula presenta ciertos problemas. El coste y mantenimiento de algunos dispositivos. Estos ocurre con las tabletas, no es un dispositivo en si costoso pero es necesario tener un por alumno. Por este motivo, hemos decido utilizar la cámara Kinect©, su coste no supera los 100€ y con una por clase es suficiente, ya que la cámara es capaz de detectar a más de un jugador.

Por otra parte, tenemos el problema del contenido. En un principio los juegos que se implementaron eran muy específicos. Un ejemplo de ello es el juego de las horas, que

hablaremos de él más adelante, que solo sirve para aprender la hora. Después de la primera entrevista con los educadores decidimos hacer juegos más genéricos, donde el contenido pueda variar pero la esencia sea la misma. Con esta idea hicimos el juego de los animales, la idea base es el juego de las parejas. Debes asociar la imagen del animal con el nombre. De esta forma si el temario cambia lo único que debemos hacer es cambiar las imágenes y los nombres.

Superando estos inconvenientes podemos conseguir que la educación sea *gamificable*, es decir, mediante el uso de videojuegos motivamos y captamos el interés del alumno hacia la asignatura. La técnica motivacional más utilizada en los videojuegos consiste en otorgar recompensas por los logros obtenidos, en este caso en el avance en el temario, entregando premios denominados *badges*.

Existen diversas plataformas educativas que buscas estos objetivos. Entre ellas tenemos MinecratfEdu (http://minecraftedu.com/). Esta plataforma utiliza el jeugo Minecraft (https://minecraft.net/) y permite la creación de mundos en los que los alumnos pueden construir diferentes elementos, así como "descubrir" zonas del terreno, cuevas, etc. Los alumnos se enfrentan a retos que les ayudan a aprender el contenido de la asignatura.

También encontramos plataformas educativas que utilizan Kinect©. KinectEDucation, su objetivo es crear una "educación conectada" mediante el desarrollo de aplicaciones, promoción e integración de estos juegos en las aulas. Se trata de exhibir la evolución de Kinect© , la exploración de los juegos en la educación, la programación en las aulas, y la integración de las actividades de Kinect© en la educación.

Otras plataformas educativas que encontramos son: *futurelab*(http://futurelab.org.uk/), su objetivo es promover la innovación en la enseñanza y el aprendizaje utilizando la tecnología. *The Education Arcade*(http://www.educationarcade.org/), buscan diseñar juegos que ayuden a un aprendizaje natural y se ajuste a las necesidades educativas de los jugadores. *Moodle* (https://moodle.org/), es una aplicación web libre que los educadores pueden usar para crear páginas web educativas, esta plataforma es utilizada por Google y Loreal para hacer la selección de personal.

Para el desarrollo de los juegos se ha empleado la arquitectura XNA que permite la creación de videojuegos mediante el lenguaje de programación C# sobre Windows. Una de las principales características de XNA es la de la simplicidad en el desarrollo de juegos, puesto que la librería XNA se encarga de los aspectos más técnicos del juego. Permitiendo que el programador solo se centre en la lógica del juego

El resto del artículo se ha estructurado en las siguientes secciones. La sección 2 describe brevemente el conjunto de juegos diseñados y sus principales funcionalidades. La sección 3 proporciona una descripción del conjunto de clases desarrolladas, y se muestra parte de la reusabilidad en el diseño que se ha tratado de seguir con objeto de poder adaptar el esquema de juego propuesto a futuros desarrollos.

2 Diseño de los Juegos

A continuación se describirán los juegos desarrollados para el sensor Kinect© y la pizarra digital. Según el dispositivo considerado (únicamente la pizarra digital y/o el sensor Kinect©) la interacción entre el jugador y el juego variará permitiendo varios tipos de juegos.

Con el sensor Kinect© se ha querido imitar el funcionamiento del ratón utilizando gestos corporales. Para indicar al juego que se desea mover el cursor debemos mover la mano, de este modo el puntero se moverá siguiendo el camino que describa la palma de la mano y para simular el clic del ratón simularemos que queremos pulsar un botón imaginario en el aire. Se trata de una interacción similar a la actualmente desplegadas en la televisiones "smart TV" de alta gama que incorporan cámaras web y software para reconocimiento de la mano.

Para las pizarras digitales, simularemos el desplazamiento del puntero deslizando uno o varios dedos sobre la superficie de la pantalla. Para simular el clic debemos poner el dedo encima del objeto que queremos clicar y pulsar sobre la zona de la pantalla donde esté dibujado.

La selección de los juegos diseñados se basan en un análisis inicial por parte de los autores del enorme conjunto de juegos educativos (tanto gratuitos como de pago) disponibles en el mercado. De este análisis se decidió definir juegos que tratasen de reforzar conceptos muy básicos como el aprendizaje de la hora, vocabulario o el aprendizaje de colores en niños. Posteriormente, y tras diversas reuniones con educadores se crearon otros juegos adaptados a una metodología educativa concreta, y que pudiese ser directamente utilizable dentro del proceso educativo.

2.1 El Juego de las Horas

La finalidad del juego es que el niño aprenda cómo se representa la hora en los relojes digitales y analógicos, mejorar la coordinación ocular-manual y la psicomotricidad fina. Para ello nos ayudamos de dos relojes virtuales: un reloj digital y otro analógico. Para este juego se han desarrollado dos versiones, una versión para el sensor Kinect© y otra para ratón.

Fig. 1. Juego de la hora, versión Kinect©.

El reloj digital indica la hora que el niño debe marcar en el reloj analógico, la hora en formato digital se representa, en formato 12 y 24 horas. Para ayudar a que el niño asocie las horas a las diferentes partes del día el fondo del juego cambia, en el ejemplo mostrado en la Figura 1, al tratarse de las cuatro de la mañana, el fondo muestra un paisaje nocturno. En la versión para Kinect©, las manecillas del reloj analógico se mueven cuando se pulsan los botones (más o menos) de la hora o los minutos respectivamente. Para comprobar si el niño ha puesto la hora correcta se debe pulsar el botón comprobar que está encima del reloj digital. En caso de acierto, o fallo, se mostrará un mensaje, en caso de error se puede volver a intentar introducir la hora, en caso de acierto se genera una nueva hora.

En la versión para pizarra digital el funcionamiento es similar, lo único que cambia es la forma de mover las manecillas. Para mover la manecillas se debe hacer clic sobre ellas y sin levantar el dedo de la pantalla arrastrar la manecilla a la posición deseada. Esta interacción permite al niño tener el control de las manecillas al "arrastrar" la horaria o el minutero hasta la posición deseada.

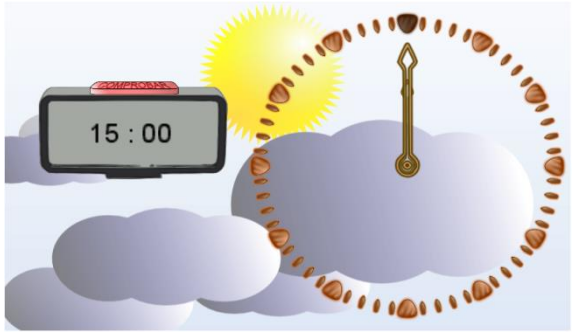


Fig. 2. Juego de la hora, versión pizarra táctil.

2.2 El Juego de las Vocales y las Consonantes

En este juego el niño aprenderá las letras del abecedario en inglés o en español, desde el menú del mismo puede elegirse el idioma. Este juego se ha diseñado principalmente pensando en el sensor Kinect©. Al inicio de cada ronda del juego el niño escuchará la pronunciación de la una letra del abecedario, en inglés o español, y deberá seleccionar la letra correcta. Cada niño tiene tres intentos para acertar la letra, si lo consigue antes de tres intentos aparece un mensaje felicitando al niño, en caso contrario aparecerá un mensaje animando al niño para que vuelva a intentarlo.

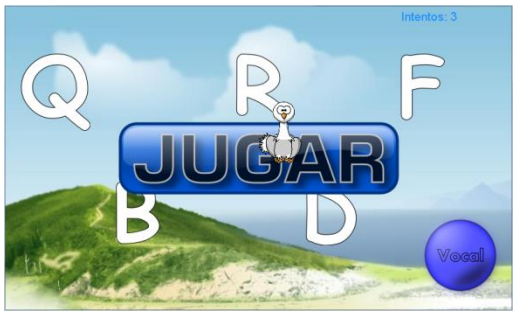


Fig. 3. Juego de las vocales y consonantes.

Este juego, como el anterior, trata de ayuda a mejorar la coordinación ócular-manual y la psicomotricidad fina. Además de ayudar en el aprendizaje audiovisual del abecedario de una forma divertida y entretenida.

2.3 Juego de Pintar

Este juego está desarrollado para interactuar con el sensor Kinect©, al igual que en el juego de las vocales y consonantes cuando empieza la ronda el niño escuchará la pronunciación del color en inglés y deberá seleccionar el color correspondiente de la paleta de colores. Hasta que no se seleccione el color correcto el jugador no podrá empezar a pintar dentro de las figuras, éstas las selecciona el juego de forma aleatoria y las que pueden aparecer son el círculo, el triángulo o el cuadrado.

Para pintar el niño deberá mover el cursor dentro de la figura y estirar el brazo. El juego calcula el porcentaje de píxeles coloreados dentro y fuera de la figura, si supera el umbral de píxeles coloreados dentro de la figura aparecerá un mensaje de felicitación y en el caso contrario aparecerá un mensaje animando al jugador a que lo intente otra vez.

Fig. 4. Juego de pintar.

2.4 Juego de los animales

El juego de los animales consiste en seleccionar la imagen que se corresponda con el nombre del animal. Se puede elegir la dificultad del juego: normal o difícil. En el nivel fácil aparecen animales que su letra inicial es diferente, y en el nivel difícil los nombres empiezan por la misma letra. Por ejemplo, en la Figura 5 se muestra un ejemplo del modo normal. En este ejemplo se muestran las imágenes de un Erizo y un Zorro, en el texto "erizo" el niño debe reconocer la primera letra, si puede realizar la correlación fonética entre el sonido de la vocal y el nombre del animal (que previamente han estudiado en el aula) identificará el animal.

Esto se debe a que los niños para aprender a leer en un principio aprenden a asociar los fonemas con las grafías. En el caso que dos palabras contengas grafías parecidas, como burro y buey, el niño no es capaz de diferenciarlas fácilmente.

El niño tutelado por un adulto, en este caso el profesor, debe de pronunciar el nombre y asociarlo correctamente con el animal.

Fig. 5. Juego de los animales.

Este juego ha sido desarrollado para interactuar con la pizarra digital o con Kinect©. Si se utiliza la pizarra digital para elegir el animal se debe pulsar sobre él con el dedo. Si el jugador acierta aparecerá una cara sonriente, y si falla una cara triste.

Con el sensor Kinect© se utilizarán posturas para ejecutar órdenes en el juego. Se levantará la mano derecha para seleccionar la imagen derecha y lo mismo con la mano izquierda.

3 Estructura de los Juegos

Para desarrollar los juegos se ha utilizado el lenguaje de programación C#, la librería XNA, la librería de Kinect© y se han implementado clases que permiten la comunicación con la Kinect© y la detección de gestos y posturas.

Las clases desarrolladas son:

- La clase `Esqueleto`. Se sincronizar con el sensor Kinect© y recibe los datos de los jugadores. Esta información se utilizará posteriormente para detectar gestos y posturas.
- La clase `GestureDetector` es la encargada de detectar los gestos.
- La clase `PostureDetector` es la encargada de detectar las posturas.

Las clases `GestureDetector` y `PostureDetector` se añaden a las listas de gestos y posturas de la clase `Esqueleto`, se comprueba si el jugador ha realizado algunas de estas acciones y se lanza el evento correspondiente.

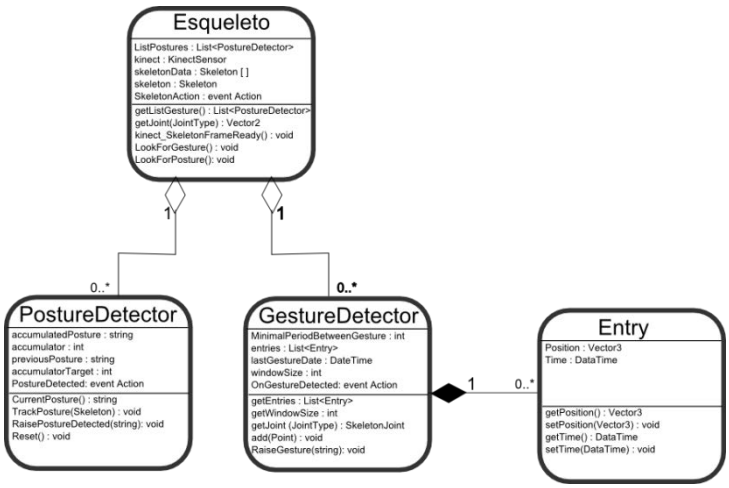


Fig. 6. Diagrama de clases.

3.1 Reutilización del juego de los Animales

Como hemos comentado anteriormente, los primeros juegos que hicimos mencionados en los aparatado 2.1, 2.2 y 2.3, son juegos con contenido específico y difícil de adaptar al temario docente.

Con la idea de poder hacer un juego que se adapte al contenido del temario se pensó en hacer una versión del juego de las parejas. Como hemos explicado en el apartado 2.4, el juego consiste en emparejar la imagen con el nombre del animal.

Actualmente la adaptación del contenido se realiza manualmente. Las imágenes se guardan en una carpeta con formato *jpg* y con el tamaño de 480x600 pixeles. Cuando

se carga el juego recorremos la carpeta de imágenes y almacenamos los nombres de las fotos en un diccionario, en la sección 3.2 hablaremos del diccionario. En esta versión del programa el nombre de la imagen corresponde con el del animal.

Por otro lado, también se puede cambiar el fondo del juego, este proceso también es manual. La imagen que hemos decidido poner como fondo debe tener como nombre *fondo. jpg* y cuando se cargue el juego se visualizará. El fondo del juego ayuda al jugador a ponerse en el contexto educativo. En este caso, como estamos con los animales el fondo es un bosque.

3.2 Nivel de dificultad del Juego de los Animales

Para modificar el nivel de dificultad en el juego de los animales descrito en la sección 2 se diseñó un algoritmo simple que permite seleccionar una etiqueta (sobre un conjunto prefijado) en función del tipo de animal que se está mostrando.

Se ha utilizado un diccionario en el cual se almacenan todas las palabras que empiezan por la misma letra en una lista (en este caso de animales). En el modo de juego "normal" se accede aleatoriamente a dos listas del diccionario, las cuales representan letras diferentes, y se escoge un animal aleatorio de cada lista. Para el modo "difícil" escogemos de una lista a dos animales (de esta forma garantizamos que cuando menos el primero de los caracteres coincidirá).

3.3 Comparativa entre la cámara Kinect© y pizarra digital

El diseño del nuevo juego (animales) permitía el aprendizaje incremental de las consonantes mediante la fonética de las sílabas. Este juego pretende lograr una asociación entre el sonido del fonema y la consonante, utilizando como método de refuerzo la imagen del animal. De esa forma, el niño puede asociar (por ejemplo) el sonido "*se*"rpiente con el animal correspondiente, e identificar que la consonante "s" con la vocal "e" corresponden a la primera sílaba del animal que indican.

Una vez concebido el juego en cuestión, se diseñaron dos versiones del mismo, la primera permitía al niño utilizar la Kinect levantando uno de los dos brazos (derecho o izquierdo) para señalar al animal que considerasen que correspondía con el texto que se les mostraba. La segunda implementación permitía al niño tocar la pizarra digital para "señalar" al animal.

El primero de los diseños (el basado en Kinect) pretende no sólo mejorar la jugabilidad, sino también enseñar otros conceptos como "izquierda" o "derecha" al niño, o incluso probar la psicomotricidad de los pequeños. El segundo trata de hacer intuitivo la interacción entre el juego propuesto y la pizarra digital, el niño únicamente tiene que "pulsar" sobre la imagen que cree que es la correcta.

En las pruebas realizadas (ver siguiente sección), se utilizó un sensor Kinect y una pizarra digital en el aula de los niños. Ellos actuaban en forma de "conferencia", es decir, sentados alrededor de la pizarra mientras el profesor o uno de los niños, seleccionaba al siguiente en el turno del juego.

3.4 Mejoras de la aplicación

Con la solución utilizada en la aplicación resolvemos el problema que teníamos en un principio, ahora la temática del juego es configurable. Con lo que ahora mismo podemos hacer varias versiones del videojuego manteniendo la misma filosofía. Es decir, en nuestro caso los niños aprenden los nombres de los animales del bosque, peros sólo cambiando las imágenes y el fondo conseguiríamos por ejemplo que los usuarios aprendieran las partes del cuerpo.

Aun así encontramos los siguientes inconvenientes que se deben solucionar en futuras mejoras:

(a) No es amigable para el docente cambiar el contenido del paquete educativo. Necesita renombrar todas las imágenes y redimensionarlas al tamaño adecuado.
(b) Como utilizamos el nombre de la imagen para hacer las etiquetas tenemos el inconveniente que no acepta caracteres especiales como guiones, la barra, los acentos, la 'ñ', etc.

Para solucionar el problema (a) se debería crear una interfaz en la cual el profesor pueda insertar la imágenes y la propia aplicación redimensiona las imágenes de forma correcta para evitar pérdida de calidad. Para el problema (b), una posible solución es que a la vez que el usuario seleccionar la imagen del juego inserte la etiquita con la cual se emparejará en el juego. De esta forma podemos insertar caracteres espaciales.

4 Evaluación y pruebas realizadas

Los juegos descritos en la sección 2 fueron instalados y mostrados a varios educadores del centro de educación infantil y primaria C.E.I.P Manuél Vázquez Montalbán (www.educa.madrid.org/web/cp.manuelvazquezmontalban.leganes/) de Leganés en Madrid.

Como se ha mencionado en la sección anterior, varios de los mismos fueron descartados, y el juego de "animales" fue finalmente seleccionado para realizar diversas pruebas con dos clases de infantil. La primera, en una clase de primero de infantil, 26 niños comprendidos entre 3 y 4 años, uno de ellos con necesidades especiales.

Este mismo juego fue modificado adaptándolo para una clase de segundo curso de educación infantil (23 niños de 4 a 5 años). En este caso se sustituyeron las imágenes por partes del cuerpo (huesos, cerebro, órganos, etc…), elementos que estaban siendo

desarrollados por la educadora. La adaptación del juego consistió únicamente en la sustitución de las correspondientes imágenes y etiquetas del texto (reutilizando el diseño de clases mostrados en la sección 3).

Uno de los primeros problemas que encontramos en la realización de las pruebas, fue la dificultad de la utilización del sensor Kinect©. El espacio de las aulas en general era reducido teniendo en cuenta el número de niños considerados, este hecho hace que se identificase más de un jugador, en particular si los mismos son inquietos o tienen dificultad para estar un largo tiempo parado. Este hecho hizo que la jugabilidad con el sensor fuese baja, al no poder capturar con facilidad el movimiento del jugador y de los "espectadores".

Por otro lado, los niños de tan corta edad todavía no tienen suficiente madurez para utilizar el sensor Kinect. No asocian el movimiento del cursor del juego con el de la mano. Les resulta más fácil e intuitivo utilizar la pizarra digital ya que sólo deben pulsar los objetos de la pantalla para interactuar con el juego.

Por lo tanto, de las cuatro pruebas realizadas (dos en cada grupo considerado), únicamente en una se utilizó la Kinect descartándola finalmente por poco usable en el entorno considerado. La utilización de la pizarra digital fue en todos los casos la más positiva al permitir una forma más simple de juego. En varios de los estudios realizados se pudo observar un comportamiento colaborativo entre los niños, algunos de ellos ayudaban a otros a identificar el animal correcto explicándoles porqué la consonante y la vocal que le seguía no podía corresponder al fonema del animal que señalaban.

5 Conclusiones y trabajo futuro

Después de las pruebas y reuniones con los educadores se realizó una serie de entrevistas con los mismos de las que se extraen dos conclusiones principales. Por una parte, los juegos inicialmente considerados y diseñados no se adecuan al temario que están impartiendo en el centro. Este hecho es particularmente significativo, dado que según la experiencia de estos educadores, es un hecho muy frecuente que la mayoría de paquetes de juegos educativos disponibles no tienen en cuenta.

Los paquetes educacionales basados en *gaming* son habitualmente de carácter general. Es decir, se diseñan sobre conceptos o metodologías docentes genéricas pero que rara vez se adaptan, o pueden adaptarse a una metodología propia, por ejemplo la utilización de fonemas para la enseñanza de la lecto-escritura.

Por otra parte, se consideró que a los niños les resultaría más fácil interactuar con el sensor Kinect, no mediante el control de un "puntero" y sus manos, sino mediante la utilización de gestos o posturas (por ejemplo, mediante el reconocimiento de la mano derecha o izquierda). Bajo esta suposición se modificó el juego de los animales para tratar de adaptarlo a un reconocimiento postural del niño. Sin embargo, el entorno altamente "ruidoso" (gran cantidad de niños pequeños en un espacio reducido) sigue

haciendo extremadamente difícil la utilización de la Kinect©. Debido a ello, la interacción con la pizarra digital es sin duda la mejor elección en el aula.

El estudio de la adaptabilidad de estos juegos mediante interfaces simples que permitiesen cambiar al docente el "aspecto" del juego es un trabajo todavía que debe ser desarrollo. En un futuro, se pretende cambiar el proceso manual que se utiliza actualmente para adaptar el contenido del juego a un proceso más automático y amigable para los profesores. Sería necesario crear una interfaz donde el docente pueda insertar la imagen y en asociarle un nombre, ya que utilizar el nombre de la imagen tiene sus propias limitaciones como hemos comentado anteriormente, en el apartado 3.2. Además tenemos que valorar el uso de una nueva librería gráfica, ya que la librería XNA dejará de tener soporte este año, una posible opción sería la librería DirectX o la librería SDL, ambos son soportadas en C#. Finalmente, debemos explorar nuevos juegos simples que pudiesen ser utilizados en otros centros.

6 Agradecimientos

Quisiéramos agradecer al colegio Manuel Vázquez Montalbán de Leganés, por permitirnos probar los juegos infantiles en la clase de infantil y muy en particular a las profesoras Cristina Morejón y Paola Weil, por su ayuda en la evaluación de los juegos y por sus consejos que han ayudado a definir el diseño final de los juegos. Por último, y muy especialmente, a todos los "peques" de las clases de primero y segundo de infantil del colegio Manuel Vázquez Montalbán de Leganés que participaron en las pruebas.

El desarrollo del trabajo aquí presentado ha sido financiado por el proyecto de investigación ABANT (TIN2010-19872).

Referencias

1. Brunner, Simon. Lalanne, Denis. Schwaller, Matthias.: *Using Microsfot Kinect Sensor To Perform Commands on Virtual Objects*, Master (Informática) Hong Kong, Université de Neuchâtel, Universitas Friburgensis, Unuversität Bern 2012. 101 h.
2. http://minecraftedu.com/
3. http://www.kinecteducation.com/
4. http://futurelab.org.uk/
5. http://www.educationarcade.org/
6. López Chamorro, Irene.: *El juego en la educación infantil y primaria*. Revista de la Educación en Extremadura. (98):, 2010.
7. Alfonso García Velázquez, Alfonso García, Josué Llull. Peñalba. *Juegos, juguetes, roles sociales y medios de comunicación* En su: *El Juego Infantil y su Metodología.* Madrid. Editex, 2009. pp. 130 - 195 (2010)
8. Cemades Ramírez, Inmaculada.: *Desarrollo de la creatividad en Educación Infantil Perspectiva constructivista*. Revista creatividad y Sociedad (12):, septiembre 2008

9. Catuhe, David.: *Programming with the Kinect for Windows Software Development Kit*. Devon Musgrave (2012)
10. Webb, Jarret. Ashley, Jame.: B*eginning Kinect Programming with the Microsoft Kinect SDK*. Jonathan Gennick. (2012)
11. Freitas, S. (2006) *Learning in immersive worlds. A review of game-based learning*. Disponilbe en: http://www.jisc.ac.uk/media/documents/programmes/elearninginnovation/gamingreport_v3.pdf
12. Freitas, S., Savill-Smith, C. and Attewell, J. (2006) *Educational games and simulations: Case Studies from Adult Learning Practice*. London: Learning and Skills Research Centre.

Implementation of a videogame: Legends of Girona

A. Rodríguez[1], R. J. García[1], J. M. García[1], M. Magdics[12], and M. Sbert[1]

[1] Universitat de Girona, Girona, Spain,
[2] Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, Hungary

Abstract. Legends of Girona is a serious game which can be used to teach local history to students in the province of Girona in Spain. The game is a graphic adventure taking place in both present-day and medieval Girona. We describe in this paper the design of the game, including aspects related to artificial intelligence, advanced computer graphics, immersive displays, exergaming and mobile platforms. We also present some directions of future work.

Keywords: serious game, history, immersive devices

1 Introduction

We present here Legends of Girona: a serious game aimed at learning the local legends of the city of Girona in Spain. The game can be used by high school students to ease retention of important happenings and dates of local history. The game is a graphics adventure which aims at providing players an immersive environment in the relevant period.

The game is a result of a collaboration between our Computer Graphics group and the TIC Education group. The storyline and plot was designed by the Education group to maximize the learning experience while maintaining the enjoyability of the game, while the Computer Graphics side used the storyline to design and implement the game.

The next section provides a historical background of the events described in the game, while section 2 describes the game architecture. The state diagram of the game can be seen in section 2.1; then section 2.2 describes the Artificial Intelligence algorithms programmed. Section 2.3 and 2.4 provide a description of photorealistic and non-photorealistic algorithms used in the game. Then section 2.5 describes the use of Microsoft Kinect to control the game. Section 2.6 describes the use of immersive display devices, and section 3 shows the adaptation of the game to run on iPhone and iPad devices. Finally, section 4 concludes the paper.

1.1 Historical Background

The game starts in present day Girona to provide a familiar setting to users, and to increase believability and identification with the main game character.

After spending some time familiarizing themselves with the city (or recognizing familiar streets in the case of players living in Girona), time travel is used to lead the player to the desired date (in our case, the siege of Girona during 1285). A similar technique has been used in the Assassin's Creed series of games [8].

We have modeled a realistic walkthrough from the Stone Bridge in Girona to the St Feliu Church along the Rambla and the Ballesteries streets. This provides a view of two landmark medieval structures, and their situation as extremes of a modern-day touristic, cultural and commercial street.

After the walkthrough, the player is transferred to Girona in the year 1285, during the siege of the city. The siege of Girona in 1285 was part of the Aragonese Crusade (1284-1285), in which the pope declared the crusade against the Aragonese king who had just conquered Sicily (a territory under papal control then). The French armies attacked the Roussillon, and besieged Girona in 1285, which was taken. A further battle saw the French defeat, with further loses due to a dysentery epidemic. The crusade ended in 1291 with the lift of the church ban on the Aragonese king.

With these historical facts as a basis, the Girona legend of the flies of St Narcis took the following form: The legend states that when the French invaders opened the St Narcis sepulcher searching for loot, a large quantity of flies exited the sepulcher instead, covering the whole city and bringing illnesses to the French soldiers, which had to retreat in haste.

1.2 Game description

The game starts in Girona's boulevard, near the tourist's office. In the office, the player is given some initial comments; then he is free to walk along the boulevard. On the other side of the boulevard is the St Feliu church, where the player is given a puzzle to solve. After solving the puzzle, the player is transferred to 1285.

The player is now outside the city and notices the siege. He must flee from the soldiers. After getting help from some citizens, he manages to enter the city. During the process, he notices that the inhabitants of Girona consider him a hero which will save the city.

Inside the walls, the player should collect some flies under the guidance of a sorceress, and search for the entrance of a labyrinth leading to St Narcis' sepulcher. The flies are to be put inside the sepulcher, so that when the soldiers open it later, the flies attack them. After the soldiers are defeated, the player is transferred back to the present.

2 Game Architecture

The game was implemented using Unity [9], which is a multiplatform game engine running on Microsoft Windows and Apple OSX which can create games for the Web, PC, OSX, IOS, Android, Flash, Xbox, PS3, and Wii platforms. Currently, we are targeting the PC, OSX and IOS platforms for our game.

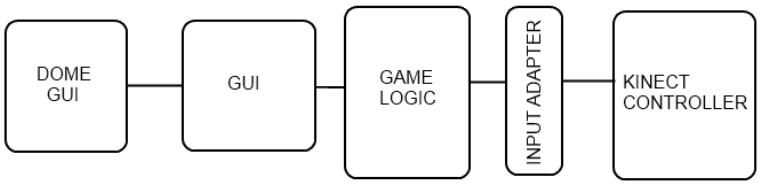


Fig. 1. Game Structure diagram

Figure 1 shows how the game class structure is divided in different blocks. The decoupling of Input and Output has been highlighted to show how this architecture can allow the use of non-conventional immersive displays (which traditionally require higher integration in the game engine) and new input devices with little effort.

The game logic block is composed of the game state and of the actors affected by the game state. There are two types of actors: friendly and hostile. A friendly actor helps us advance in the story and interacting with them changes the game state. Hostile actors produce a penalty when we interact with them. The next section describes the game states.

2.1 Game levels and State diagram

The game is structured in six levels:

1 Current-day Girona
2 The St. Feliu Church
3 Outside of the city walls during the 1285 Siege
4 The city wall gate puzzle
5 Inside of the city walls
6 The labyrinth

Screenshots of the different levels can be seen in figure 2. A description of the state diagram shown in figure 3 follows.

Current-day Girona Current-day Girona is divided in two states: the initial state, in which movement is limited to a five meter radius of the original position, and which only allows interaction with the tourist office.

After interaction, the second state allows the user to move along the streets leading to the St Feliu Church. Entering the church loads the second level.

The St. Feliu Church This level has four states.

1 In the initial state we may either find a key or a door.
2 The key produces a change in state prompting the user to search for the door.

Fig. 2. Levels of the Legends of Girona game

3 Seeing the door without the key will prompt the user to search for the key.
4 After the key has been found in state 3, the user is prompted to return to the door.

In states 2 and 4, the door may be activated to load the third level.

Outside of the city walls during the 1285 Siege This level is the most complex one, with eight states which include an optional intermediate sub-mission.

1 In the initial state we see how the protagonist wonders where he is and concludes that he needs to search for help. After the initial monologue is ended, there is an automatic transition to the next state
2 We are outside of the city of Girona surrounded by guards to avoid and with the objective of finding help. The player is shown in a first-person perspective. To change state, we must find the Merchant and ask for help, after which state 3 is achieved.
3 In this state we wear medieval clothes to be less conspicuous and a pendant marking us as the chosen one. A third person perspective is used to ease navigation in the city outskirts. The objective is to talk to a lame man in the city while avoiding the guards. Once we talk to the man state we reach state four.

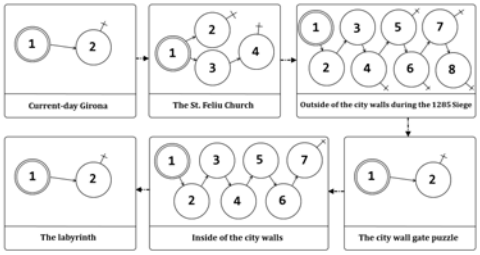


Fig. 3. State diagram of the Legends of Girona game

4 When entering state 4 we see night fall on Girona and the lame man has told us how to enter inside the city walls. From this moment we may go to the city gates and that will induce the load of the next level. If we want the walk to the gates to be easier, a sub-mission may be done so that the lame man helps us distract the guards. The sub-mission consists in getting 500 coins for the lame man. To change the state, we can go and talk to the Merchant, which triggers state 5.
5 In this state we have a package given to us by the Merchant. We may take the package to the blacksmiths house so that the Merchant gives us the coins we need. After delivering the coins, state 6 is entered.
6 In this state we need to tell the Merchant that the package has been delivered. Once we do, we receive the 500 coins and enter state 7.
7 In this state we have the coins and giving them to the lame man ends the sub-mission, leading to state 8.
8 The lame man distracts most of the French guards which were patrolling outside of the city wall. We only need to go to the city gates to change to the next level.

The city wall gate puzzle This level has two states, the initial one until the year of the siege is provided by the user, and a final state to provide the transition to the next level.

Inside of the city walls This state has seven states, but without optional sub-missions.

1 We start in the inside of the city walls and must investigate to know what to do. After talking to a boy near the gates, state 2 is reached.
2 In this state, our purpose is clearer. We need to search for a sorceress and request her help. After crossing the city to find her and talking to her, state 3 is triggered.
3 Now we need to use a basket (given to us by the sorceress) to collect a series of flies which are located all around the city. The higher the difficulty level,

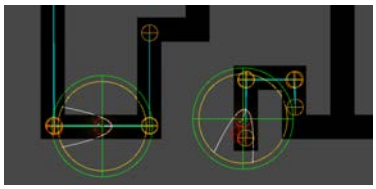


Fig. 4. Pathfinding and detection zones for soldiers

the more flies we need to find and the less time we have to complete the task. If the task is complete, state five is triggered; otherwise state four will occur.

4 Time is over, and the flies have escaped the basket, breaking it. We need to talk to the sorceress again to get another basket, after which we reach state 3 again.

5 The basket is full of flies, so we need to go back to the sorceress' place. After talking to her, she gives us a hint and state six is reached.

6 We now need to talk to a Jew to get the labyrinth's key. After talking to him the transition to the 7^{th} state is triggered.

7 We have the flies, the hint and the key, so when we reach the labyrinth door the next level is loaded.

The labyrinth This level also has two states, the initial one and a final one with the end of labyrinth animation, which leads to the end of the game.

2.2 Artificial Intelligence

The control of the behavior of the non-player characters is based on artificial intelligence rules to predict and anticipate the player [6]. This is the most relevant aspect in the city guard characters, which follow the player through medieval Girona.

Three aspects need to be taken into account:

– **Pathfinding algorithms to patrol and move** All soldiers and some scripts which will be applied in the future to simulate the population of both current-day and medieval Girona use the standard pathfinding integrated in the Unity 3 engine to calculate the path to their destination. The colliders of the objects and the terrain of the scene are used to create a walkability map; this map can be used to calculate a path to any point, as long as the point is contained in the map. Furthermore, it includes a waypoint system so that soldiers patrol in the zones delimited by them. Soldiers will decide which waypoint to visit next using a probabilistic function. To simulate more characters, waypoints which are not visible for the player, and which allow NPC to appear and disappear also exist. Figure 4 provides an overview.

– **Detection algorithm** All soldiers have a detection algorithm to find objectives in a circular detection zone. Additionally, a vision zone exists, modeled by a parabola in the (x,z) plane. When an enemy enters the intersection of both zones, the soldier will prosecute the enemy. The player has two ways to flee: move farther than a predefined distance of the guard, or moving out of sight by hiding behind a building. If the guard cannot see the player, he will go to the last point the player was visible from the guard position and search around randomly for a few meters. If the player stays out of view, the guard will find the closest point in its patrol area and return there.
– **Prediction and anticipation algorithm** This algorithm uses the last movement vector of the player to calculate the future position of the player, so that the guard tries to intercept the player at that point. This is especially effective for soft turns (easily anticipatable) and hard turns, as this reduces the player speed and allows the guard to close in.

Additionally, all behaviors are affected by the difficulty level, thus:

– **Pathfinding algorithms to patrol and move** The patrol and persecution speeds are affected. At high difficulty levels, the guard and the player run at the same speed, so the user cannot exit the prosecution area and needs to hide.
– **Detection algorithm** The detection and prosecution radii are increased in high difficulty levels and decreased in low difficulty levels. The latus rectum of the vision field parabola (defining the amount of periferic vision) is also increased or decreased depending on the difficulty level.
– **Prediction and anticipation algorithm** The prediction distance for the guard is changed, using optimal levels for high difficulty levels and disabling the feature for low difficulties.

We have requested users' impressions on the behavior of the guards, and the response has been positive. These easy to implement procedures create a behavior for the guards which the users of the game consider realistic.

2.3 Realistic Graphic Effects

The realism in a video game contributes to game immersion and enjoyability. We are using the GameTools routines [2] to produce realistic materials such as metals, including pseudo-raytracing reflections, in realtime and with low computational cost.

The routines work by creating and updating distance and color cube maps centered at the objects with the GameTools materials, and using a GPU shader to evaluate the correct pixel color using a binary search on the distance cube map followed by a texture access to the color cube map. The use of this technique provides realistic images even when the maps are only updated rarely, providing highly realistic images with low computational cost. The slight decrease in fps does not affect gameplay. These routines can be integrated easily in already existing games using drag-and-drop of templates to create the necessary environment maps. The resulting effects can be seen in figure 5.

Fig. 5. Realistic reflections using GameTools effects

2.4 Stylized Rendering

In certain cases such as historical storytelling, artistic stylization may provide greater experience and a more memorable atmosphere then realistic depiction. We adapted a collection of screen-space Non-Photorealistic Rendering (NPR) techniques to Unity, enabling game designers to easily change the rendering style of the game. We designed our effects based on popular techniques from visual arts, especially comics. A common characteristics of comics is simplified depiction with large flat surfaces and minimal texture details, while enhancing important information such as character contours. In order imitate this style, we implemented texture simplification methods and luminance quantization based on [11, 3] to remove unwanted details, and the artistic edge detection of Winnemöller et al. [10] to enhance important ones. Changing the color of rendered shadows can create a variety of effects, such as impressionism or chiaroscuro [7]. Shadowed pixels are determined by rendering the image once with and once without shadows and comparing the two results as a post processing, shadow color is changed using the obtained mask. To enhance the illusion of depth, we provide depth varying detail level of textures, edge thickness and color saturation, each technique commonly used by painters and comic drawers. The atmosphere of the rendered images can be changed by adapting the color palette to a given example image, as in Reinhard et al. [5]. Most of the effects can be used with no measurable effect on speed. At Full HD resolutions, the Abstraction effect can reduce fps around 25 %, while shadow effects are the most expensive, at 33 % reduction in frame rate. All our effects are applied to a camera object as a post-processing filter [4], permitting them to be used in any games. Figure 6 illustrates several styles created by our implementation.

2.5 Kinect

In addition to realistic graphics and game physics, functional realism can greatly enhance immersion in the virtual world. Natural User Interfaces (NUIs) allow users to perform different actions in the game as they would do so in the real world. We have integrated the Microsoft Kinect into our game, and implemented a Unity package based on the official OpenNI wrapper for Unity along and the

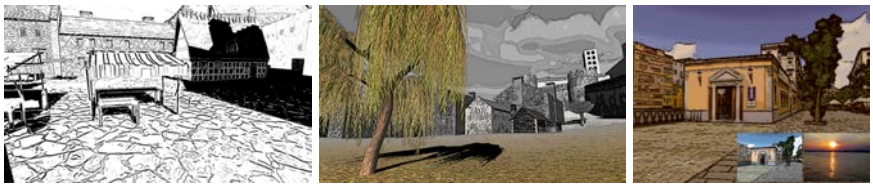


Fig. 6. Stylized renderings of Legends of Girona (left to right): black and white, depth-based desaturation, color style transfer.

NITE middleware that supports control by user gestures. The package contains a compact Kinect control object dealing with user selection, skeleton extraction and gesture tracking and providing a selection of gestures that can be added to any scene using drag-and-drop. The Kinect control integrates seamlessly into the system, it works in conjunction with the standard mouse-keyboard input without any modification of the underlying game logic.

Game control in Legends of Girona consists of navigation (moving, rotating and jumping), interaction with game objects and help (world map and quest log). Our implementation is an "exergame" requiring the user to mimic the real world actions, for example, walking in place toggles the avatars forward movement, jumping makes the avatar jump and opening the map requires the arms to be spread (similarly to holding a large map). Figure 7 shows examples for the different gestures in the game. The performance of the game was not significantly affected by the use of Kinect.

Fig. 7. Examples of Kinect gestures (left to right): jump, walk, open map.

2.6 Immersive displays

Immersive displays surround the player and provide a better viewing experience. Figure 8 shows the devices we support. We have developed a unity package which can integrate easily in existing in games and add the required functionality to display in immersive displays. The package consists of a cubemap-generating

game object and a set of post-processing GPU shaders to generate the final image (some details on the internal structure of the package can be seen in [1]). In a 3.1 GHz Intel Core i5 Mac with 8 GB of RAM and an AMD Radeon HD 6970M with 1GB of RAM, the game can be rendered at 200 fps in a flat monitor, while the Dome and immersapod visualizations render at 66 fps. The reduction in performance is compensated by the increase in realism, and in any case does not impede gameplay.

The result of the shader for Dome devices can be seen in figure 9 (left). The corresponding images for an immersapod device can be seen in figure 9 (right). The deformed images shown here correspond exactly to the inverse of the transform provided by the camera lens in these devices, so the final image shown to the user is an undistorted view of the scene with a large field of view.

Fig. 8. Dome (external and internal views) and immersapod.

3 IOS Port

To port the Legends of Girona game to the iPad 2 hardware, some things had to be modified and adapted to ensure correct performance. In particular, the polygonal load, the draw calls and the memory load had to be reduced. Additionally, specific controllers were designed. The game was ported during a period of three weeks by a single developer.

The polygonal load in the scenes was reduced by simplifying some of the models in the current-day Girona level. The decorations in the Rambla street have been remodeled, and the tree models have been simplified. Level three (outside of the city walls) required similar changes, plus a limitation on the number of soldiers. Furthermore, night scenes suffered from an excessive number of lights (the torches). Although Unity will disable non-visible lights, a more stringent method was developed and programmed to ensure that only the most nearby torches are active.

To reduce the memory load, in order to allow execution on the iPad 2, the resolution of all game textures was lowered, and the game audio was changed to use streaming from disk, alleviating the memory pressure. The current-day Girona level required more drastic measures: the scene was changed to take place during the evening, in order to justify a decrease in visibility. The far

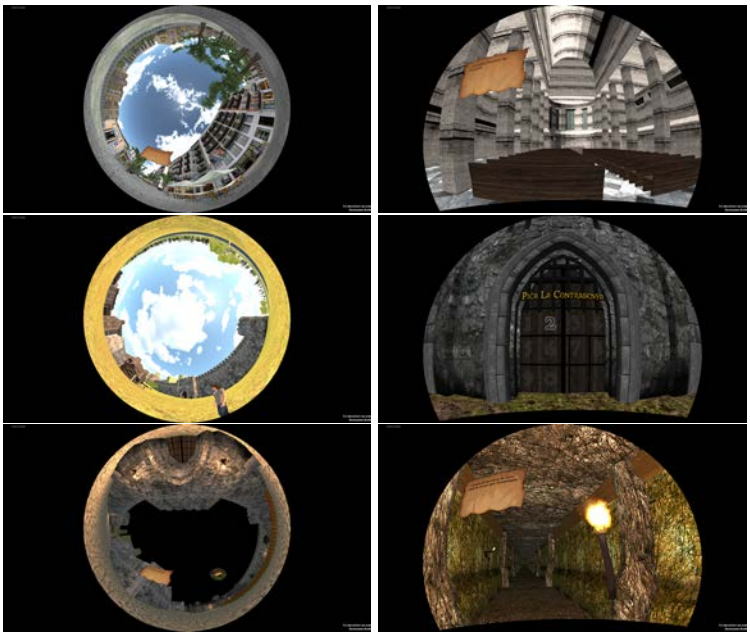


Fig. 9. Output of the shader for visualization on a dome (left) and immersapod (right)

plane of the camera was reduced to a quarter of the original distance, and black fog was added to hide the transition. The number of loaded textures was thus sufficiently reduced to fit the whole scene in memory. However, during scene transitions both scenes are loaded, so in order to avoid this (which would put the memory use above the permitted limits) a new, empty scene was added to enable the unloading of all the resources from the previous level before loading the next one.

To adapt the controls, the solution chosen was to add two simulated analog controls, one on the bottom right corner and another one in the bottom left corner of the touch screen. The right one controls the movement of the player while the left one controls camera rotation. To ease gameplay, the actions to take and activate objects, and to talk with other characters, are now performed automatically when the player comes in close vicinity to them. The gameplay has also been changed to use a first-person perspective in all scenes. The game performance is 25-30 fps, so playability is conserved.

4 Conclusions and future work

We have shown the detailed architecture and design of the Legends of Girona game. The main purpose of this serious game is teaching the history and legends of the city of Girona to high school students, while providing an enjoyable

experience. We have shown how advanced graphic effects and new interaction devices can be included in a straight manner into the game, and how to port the game to mobile architectures.

Our future work will focus in two different aspects: a) creating new content to show other historical periods and legends, and b) extend and abstract our libraries so that more graphic techniques (both photorealistic and non-photorealistic effects) and other interaction devices and techniques can be integrated seamlessly into games.

5 Acknowledgements

This work has been supported by the research projects coded TIN2010-21089-C03-01, IPT-2011-1516-370000 and IPT-2011-0885-430000 (Spanish Commission for Science and Technology), and by grant 2009SGR643 (Catalan Government).

References

1. García, R., Magdics, M., Rodríguez, A., Sbert, M.: Modifying a game interface to take advantage of advanced I/O devices: a case study. In: Proceedings of the 2013 International Conference on Information Science and Technology Application (2013)
2. García, R.J., Gumbau, J., Szirmay-Kalos, L., Sbert, M.: Updated gametools: Libraries for easier advanced graphics in serious gaming. In: Asian-European Workshop on Serious Game and Simulation, 25th Annual Conference on Computer Animation and Social Agents (CASA 2012). Nanyang Technological University (2012)
3. Kyprianidis, J.E., Döllner, J.: Image abstraction by structure adaptive filtering. In: Proc. EG UK Theory and Practice of Computer Graphics. pp. 51–58 (2008)
4. Magdics, M., Sauvaget, C., Garcia, R., Sbert, M.: Post-processing NPR effects for video games: a case study. Poster at Expressive 2013 Conference (2013)
5. Reinhard, E., Ashikhmin, M., Gooch, B., Shirley, P.: Color transfer between images. IEEE Comput. Graph. Appl. 21(5), 34–41 (Sep 2001)
6. Rich, E., Knight, K.: Artificial intelligence (2. ed.). McGraw-Hill (1991)
7. Sauvaget, C., Boyer, V.: Stylization of lighting effects for images. In: Signal-Image Technology and Internet-Based Systems (SITIS), 2010 Sixth International Conference on. pp. 43–50 (2010)
8. Assassin's Creed 4 Black Flag — Official GB Site — Ubisoft. "`http://assassinscreed.ubi.com/ac3/en-GB/index.aspx`" (2013), accessed 13 May 2013
9. Unity Technologies: Unity - Game Engine. "`http://www.unity3d.com/`" (2013), accessed 18 February 2013
10. Winnemöller, H.: XDoG: advanced image stylization with eXtended Difference-of-Gaussians. In: Collomosse, J.P., Asente, P., Spencer, S.N. (eds.) NPAR. pp. 147–156. ACM (2011), `http://dblp.uni-trier.de/db/conf/npar/npar2011.html`
11. Winnemöller, H., Olsen, S.C., Gooch, B.: Real-time video abstraction. ACM Trans. Graph. 25(3), 1221–1226 (Jul 2006)

Drum-hitting gesture recognition and prediction system using Kinect

Alejandro Rosa-Pujazón, Isabel Barbancho, Lorenzo J. Tardón, and Ana M. Barbancho

Dpt. Ingeniería de Comunicaciones, E.T.S.I. Telecomunicación,
Universidad de Málaga, Campus de Teatinos s/n, 29071 Málaga, Spain
alejandror@uma.es,ibp@ic.uma.es,lorenzo@ic.uma.es,abp@ic.uma.es
http://www.atic.uma.es/

Abstract. This paper presents a gesture-based interaction technique for the implementation of a virtual drum using a 3D camera-based sensor. In particular, a human-computer interface has been developed to recognize drum-hitting gestures using a Microsoft Kinect device. The system's main purpose was to analyze and minimize the effects of the delay between user's input commands and the execution of the command by the system. Two implementations are discussed: one based on Machine Learning techniques for gesture recognition, and the other one based on a signal processing approach, using a Wiener lineal predictor. Both implementations showed to be able to predict user movements to compensate for the lag introduced by the sensor device.

Keywords: Music Interaction, Gesture Detection, Human-Computer Interaction, Machine Learning

1 Introduction

Music interaction interfaces are usually confined to the use of the traditional musical instruments. Yet, in general terms, the mechanics and abstract concepts of music are not usually known to most lay people. Furthermore, in order to learn or understand the different aspects of music theory it is necessary to devote a considerable amount of time to such purpose. However, the evolution of sensing and motion-tracking technologies has allowed for the development of new and innovative human-computer interfaces that have changed the way in which users interact with computer applications, thus offering a more 'natural' experience than the one had with a more conventional setting, which can also help to lower the barriers of the inherent abstract nature of musical concepts.

Advanced human-computer interfaces to implement a more natural or immersive interaction with music have been proposed and/or studied in previous works [3] for a wide array of applications: gaming [8][27], new instruments creation/simulation [12], medical rehabilitation [6], modification of visual patterns by using sung or speech voice [15], body motion to sound mapping [1][5][9][14],

orchestra conductor simulation [18][21][25], tangible and haptic instrument simulation [2][10], drum-hitting simulation[11][19][26][20], etc.

Some of the problems most commonly identified with advanced human-computer interfaces is that they are usually expensive, intrusive and/or bulky, being prone to raise ergonomic issues. Fortunately, the emergence of devices like the Wiimote and Kinect have helped to mitigate such issues. Off-the-shelf devices have also been studied for the creation of new interaction models for music performance, such as the previously mentioned Wiimote [22] and Kinect [17][25][20][29][24] devices, and even regular mobile phones and smartphones [7].

In this paper, we aim to present an advanced human-computer interface for drum-hitting gesture recognition based on a Kinect sensor. As previously indicated, the Kinect sensor is mostly inexpensive and does not hinder user movements in any way, thus being completely non-intrusive. However, prior works [3][20][16] have shown that the camera sensor introduces a meaningful lag that, along with whatever delay is introduced by the application itself, may be noticeable enough to diminish the quality of the overall experience; concretely, while the Kinect sensor works at a nominal rate of 30 frames per second, the latency can reach values in the order of roughly 0.1-0.3 seconds [16]. Therefore, we propose in this paper a system that improves upon the approach followed in [3] and takes into consideration such delay and tries to compensate its effect by predicting/anticipating user's drum-hitting gestures.

2 Methods and Materials

Due to its lack of intrusiveness and its relatively low-cost, off-the-shelf nature, we opted for a Microsoft Kinect for XBOX device in our human-computer interface design. Since the Kinect 3D sensor actually provides coordinates for each position joint within its field of view, it could be possible to define the position and dimensions of a virtual drum within that space. A first iteration of a virtual drum simulator using this device was implemented, defining an area of the physical space around the user that was associated to the virtual drum volume. Additionally, the application rendered a virtual environment using C/C++, OpenGL graphics library [23], and the OGRE graphics engine [13], so that the user had a visual reference in order to "hit" the virtual drum (see Fig. 1). The sounds were read from wav files, using the OpenAL audio library. The human-computer interaction interface was implemented with the OpenNI library and the NITE plugin, using full-body skeleton tracking to track the motion of both hands at the same time. The skeleton node corresponding to the head was used as a reference to place the virtual objects in the environment (i.e. the virtual drum).

2.1 Volume-based virtual drum

As previously indicated, an initial implementation of the virtual drum was performed by detecting the position of the user hands relative to a space volume corresponding to the virtual drum, i.e. a sound will be played whenever the user

'hits' the volume defined. We want the system to be usable without tools of any kind, so that the user can play with his/her hands alone. Thus, instead of tracking sticks, mallets or tools of any kind, the drum is played with hands alone (sort of like a tan-tan). Such implementation also helps to address the fact that the technology used cannot accurately detect moves that are too short (i.e. just a short flick of the wrist). Following a tan-tan interaction metaphor, users are more likely to perform large enough gestures than using sticks or similar tools.

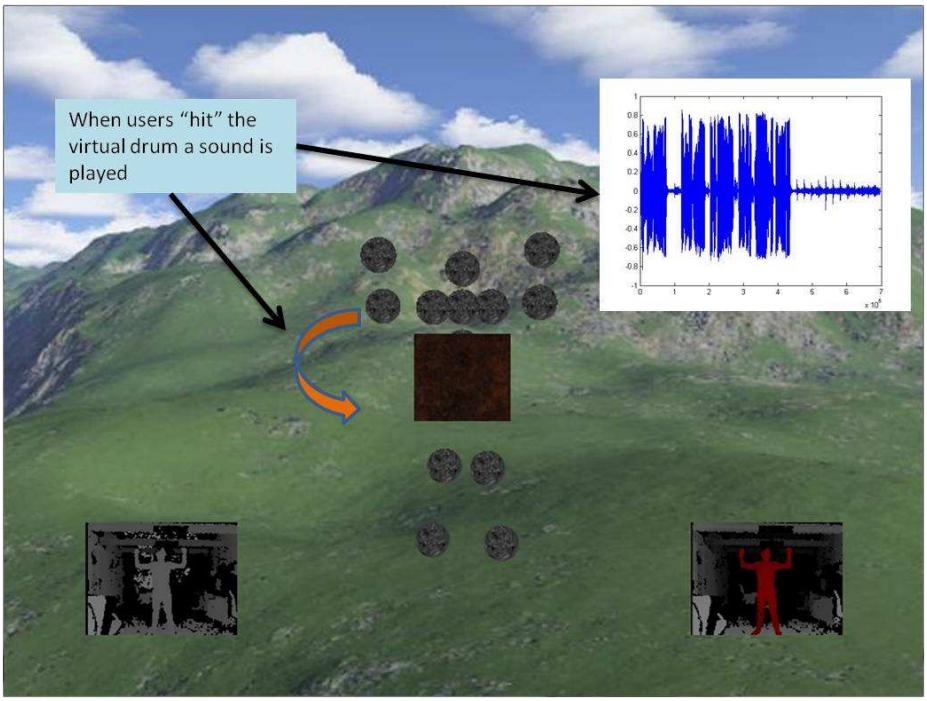


Fig. 1. Virtual Environment

The purpose of this application was to serve as a pilot study to observe potential key features in the drum-hitting gestures performed by users, as well as testing the impact of the delay in the human-computer interface on the overall experience of the users. The lag introduced was measured by recording the user performances, using a EOS 5D Mark II camera. By combining the analysis of the recordings performed and the actual times when the drum-hitting event was triggered, it was found that the overall lag in our application was approximately of 220 ms (the application ran at approximately 20-25 fps, so the lag was roughly 5 rendering frames, fig. 2). Thus, the lag introduced by the system is high enough to have an important impact in the overall experience. Specially, the lag induced becomes particularly relevant in the case of swift movements, as is the case of a drum-hitting gesture (see fig. 2).

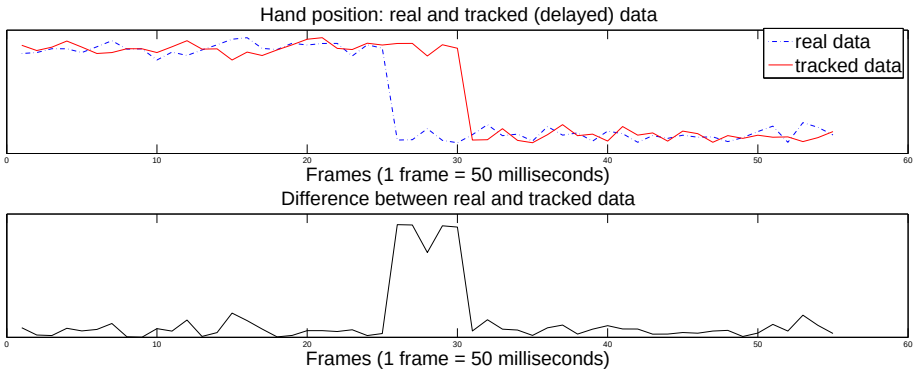


Fig. 2. Real data and tracked data: effects of the delay

2.2 Gesture recognition based on feature analysis

As found in the pilot study, the delay between user gesticulation and the actual response of the system in our application took too long for an adequate feeling of drum-hitting, hindering the purpose of the system as a virtual drum simulator. In this subsection and in the next one, two different ways to address this problematic are presented. Instead of trying to define some volumetric space around the user to "place" the virtual drum, we resort now to a more gesture-driven approach to implement the drum-hitting events. Therefore, whenever a drum-hitting gesture is detected, the system will respond by playing the corresponding sound.

In order to properly detect a given gesture, an analysis to find the most adequate features was performed. From this analysis, we identified that the most deciding features of a drum hitting gesture would be the velocity and acceleration over the vertical axis; indeed, it stands to reason to expect peaks of speed/acceleration whenever the user intends to hit the drum and, at the same time, downward movement that is not a positive gesture would be performed at lower speed and acceleration levels. In order to account for the fact that different users will have different sizes and that the magnitude of both the acceleration and the velocity will depend on the size of the user's tracked skeleton (dependant also of the position of the user along the Z axis), the data was normalized by a factor calculated as indicated by the following equations, using the coordinates of user's head and torso as a reference.

$$NormVel = \frac{Vel}{HeadPos - TorsoPos} \tag{1}$$

$$NormAcc = \frac{Acc}{HeadPos - TorsoPos} \tag{2}$$

For the detection of the gestures performed, we applied Machine Learning techniques to classify whether the user has performed a gesture or not. In particular, a logistic regression model [4] was trained to perform such detection.

The logistic regression model is characterized by a set of weights for each of the features considered (Θ_i) and a regularization parameter λ, in order to minimize the cost function $J(\Theta)$, as per the following equations (x_i are the values for each of the features considered, and y is the resulting classification value):

$$z = e^{-(\Theta_0 + \sum \Theta_i x_i)} \tag{3}$$

$$H_\Theta = \frac{1}{1+z} \tag{4}$$

$$if H_\Theta >= 0.5, \text{ then } y = 1 \tag{5}$$

$$if H_\Theta < 0.5, \text{ then } y = 0 \tag{6}$$

$$J(\Theta) = \sum (H_\Theta - y)^2 + \lambda \sum \Theta_i^2 \tag{7}$$

Training data was collected with the help of 10 participants, who were instructed to perform both positive gestures (actual drum-hitting gestures) and negative gestures (tracked motion that was not intended as a drum-hitting gesture). All the gestures performed were done with slight variations, to account for the variability that these gestures might had in a real scenario. Position, velocity and acceleration of hand movements were tracked and normalized as per equations 1 and 2 to minimize the effects of size. Once collected, the data was further processed to extract only the data corresponding to downward movements, and to eliminate the effects of noisy samples because of imperfections in the measures, the data tracked was further pruned by selecting only those gestures performed at a speed of at least 0.1 normalized units per second. The gestures were then identified by looking for the start and end of downward movements (negative velocity). An example of such refined data can be found in figure 3

For each gesture, the peak (maximum) values of downward velocity and downward acceleration were extracted as features for the logistic regression classifier. Thus, the classifying criteria followed the equations below.

$$z = e^{-(\Theta_0 + PeakVel * \Theta_1 + PeakAcc * \Theta_2)} \tag{8}$$

$$H_\Theta = \frac{1}{1+z} \tag{9}$$

$$if H_\Theta >= 0.5, \text{ then } y = 1 \tag{10}$$

$$if H_\Theta < 0.5, \text{ then } y = 0 \tag{11}$$

The value of the variable y will be 1 whenever a gesture is correctly classified as a positive gesture (drum-hitting gesture) and 0 otherwise (negative gesture).

The classifier was trained with a total of 440 gestures (273 positive and 167 negative) identified from the data collected. In addition to the training data, a set of cross-validation data was used to regularize the logistic regression, with a total of 144 samples (73 positive and 71 negative). Both features (peak velocity and peak acceleration) were scaled accordingly (by calculating the mean and variance of the training set). The resulting data for the Θ parameters of the model and the regularization parameter λ are summarized in table 2.2. The success rate with the cross-validation data was of 93.06 %.

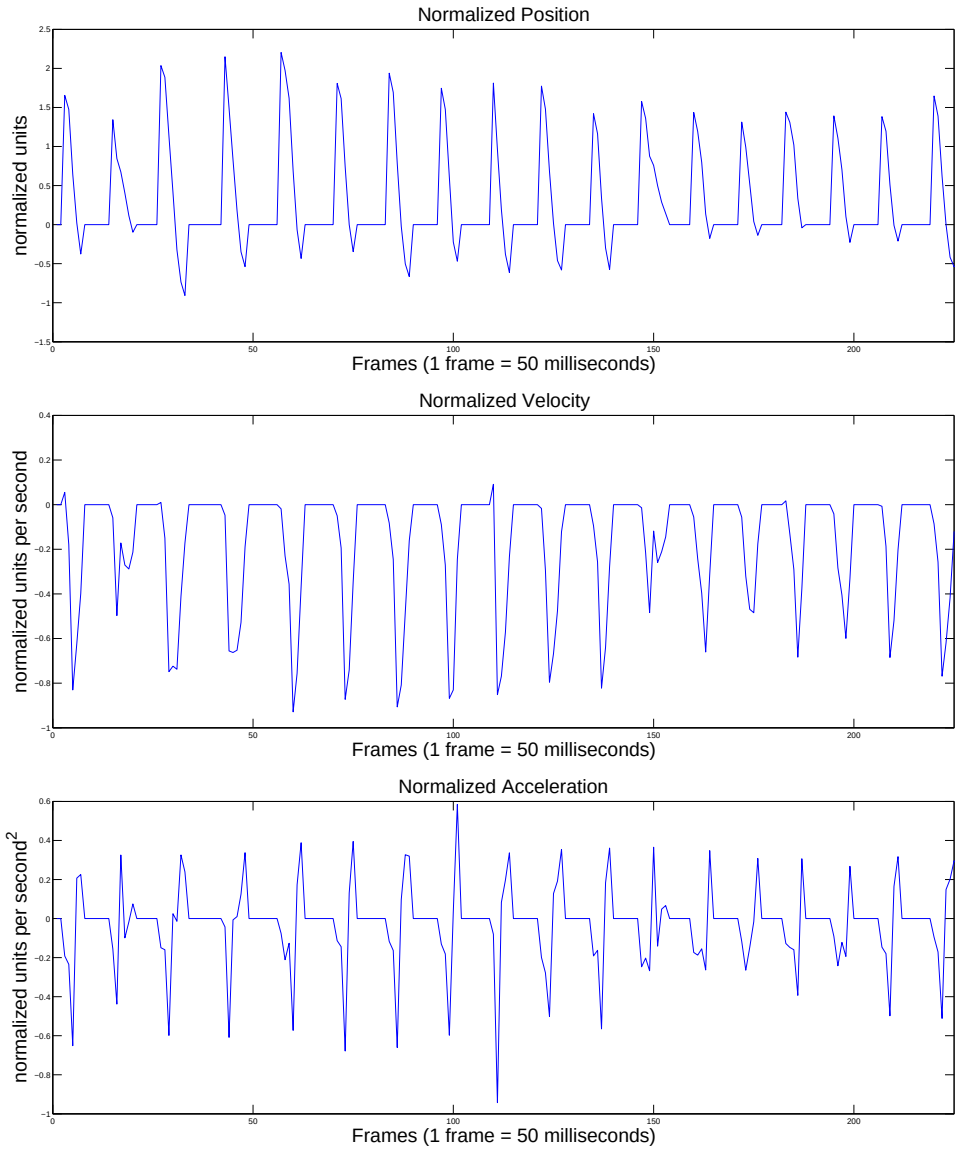


Fig. 3. Samples of extracted normalized data

Θ_0	Θ_1	Θ_2	λ
0.9221	-1.7931	-0.6101	10

Table 1. Parameters of the logistic regression model for the first system

2.3 Forecasting based on a Wiener linear predictor

As previously indicated, it was found that the delay between user interaction and the generation of sounds was as high as 5 rendering frames. Since the application obtains one motion sample for each hand per rendering frame, that means that the system is tracking motion with a delay of 5 samples. In order to address this problem, it is necessary to predict the instant when the user performs a drum-hitting gesture. This situation was addressed by using a different, alternative implementation of our gesture recognition system. In this case, we integrated a linear predictor based on Wiener filtering [28] to forecast the position of future samples according to the previously tracked motion data. Wiener filters are based on the least minimum mean square error (LMMSE) estimator; in particular, the mathematical model for the Nth-order Wiener linear filter to predict the l-th future sample of a signal $x[n]$ follows the equation:

$$\begin{bmatrix} R_{xx}[0] & R_{xx}[1] & \dots R_{xx}[N-1] \\ R_{xx}[1] & R_{xx}[0] & \dots R_{xx}[N-2] \\ \vdots & \vdots & \vdots \\ R_{xx}[N-1] & R_{xx}[N-2] & \dots \quad R_{xx}[0] \end{bmatrix} \begin{bmatrix} h[1] \\ h[2] \\ \vdots \\ h[N] \end{bmatrix} = \begin{bmatrix} R_{xx}[l] \\ R_{xx}[l+1] \\ \vdots \\ R_{xx}[l+N-1] \end{bmatrix}$$

where $R_{xx}[n]$ is the autocorrelation of the input signal. In particular, we used a Wiener filter to predict l=5 samples to cover the delay in our application, and the order of the filter was set to N=30.

Θ_0	Θ_1	Θ_2	Θ_3	Θ_4	Θ_5	λ
-0.8837	-1.7111	4.0256	4.8970	3.2626	-0.6958	1

Table 2. Parameters of the logistic regression model for the Wiener scheme system

The Wiener filter was applied to our data set to extract estimates of future predicted positions as a reference for our gesture detector. More specifically, the data set was processed in a similar fashion to the previously presented system, isolating gestures corresponding to downward movements with a downwards speed of at least 0.1 normalized units per second. In this case, however, the different gestures identified were combined to form a single mathematical signal, which the Wiener filter was to be applied onto. We used five different Wiener filters, to predict respectively the l=1,2,...5-th samples, and the 5 predicted samples were then used as features for a machine learning process regarding once more a logistic regression classifier [4]. By using these 5 samples as features of our learning process, the system could predict the future shape (in 5 samples time) of the signal from the data collected in the last 30 samples corresponding to downwards movement. This way, we expect the system to be more robust against false positives.

The classifier was trained this time with a total of 666 gestures (259 positive and 407 negative) identified from the data collected. In addition to the training data, a set of cross-validation data was used to regularize the logistic regression, with a total of 186 samples (66 positive and 120 negative). The five features considered (predicted samples at times l=1,2,..5) were scaled accordingly. The resulting data for the Θ parameters of the model and the regularization parameter λ are summarized in Table 2.3. Success rate in this case for the cross-validation data was of 93.01%.

3 Discussion

The work presented here has focused on the implementation of a natural, hands-free interaction paradigm for musical performance simulating a drumkit. Specially, the main focus has been on minimizing the lag introduced. Previous works have focused on using autocorrelation-based predictions to prevent this effect [26], yet this assumes a certain periodicity in the user performance. In this paper, we propose two machine-learning process which addresses this issue without such constraints.

The first system is based on analysing acceleration and velocity features to detect whenever the user intended on "hitting the drum". The main advantage of this system is that it does not require the computer to previously acquire any kind of data to properly detect user gestures. Instead, whenever a pair of acceleration and velocity peaks high enough were detected, a drum-hitting gesture is recognized, thus triggering the corresponding event to play an appropriate sound at the correct time, using the prediction scheme described. In other words, when the user performs a hitting-like gesture such that his/her hands draw a large arch, the system will most likely recognize the gesture before the user finishes his/her motion, thus compensating for the lag introduced by the sensing system. Nevertheless, if the user's motion draws a shorter arch (like a short motion with a flick of the wrist), the system might not detect the expected peaks of a hitting gesture.

The second system is designed more specifically to deal with the issue of the delay introduced. Thus, by using Wiener filters to implement predictors of up to 5 samples, it is possible to forecast how the motion will evolve in an interval of 250 milliseconds, which easily covers the lag found in the application. The disadvantage of this system is that the gesture recognition depends on previously recorded positive gestures, so the user will need to perform several gestures for the system to learn before the performance itself can be started.

Both classifiers were further tested using a set of 100 tests. The results provided showed a successful recognition rate of 98% for the first system, and 96% for the Wiener prediction system. From observation, it was found that the system failed to recognize gestures when the downward motion was too short, i.e. when the gesture was little more than a flick of the wrist.

Both schemes were tested by 20 different users, as well as a drummer. Every participant was specifically asked to indicate if they had noticed a delay in

the system response. They tried first the originally designed virtual environment with the volume-based detection, then the first system proposed, and then the one based on Wiener predictors. In the case of the volume-based detection system, all the users noticed the lag in the system response (as expected). When using the first gesture recognition system, 17 participants declared that they noticed no lag in their performance; only the drummer noted that he appreciated some lag in the performance, but he remarked that he found it mostly meaningless. When testing the system based on the Wiener predictor, again none of the users found any lag in the performance, yet they all appreciated that the system was responding better than with the previous gesture detection system. Specially, the drummer perceived no lag at all in this case. A video showing the drumkit simulator running the first system implementation can be found at http://youtu.be/fysJyy0D434

4 Conclusions and Future Works

In this paper, two alternatives have been proposed to implement a drum-hitting gesture recognition system for real-time performance. The first system addresses this issue by detecting peaks in the downward velocity and acceleration, while the second system uses a five-samples-prediction to minimize the effects of the lag. After testing, both systems have been shown to adequately address the problem of lag perception, and these results were corroborated by a subjective evaluation with a limited number of participants. Future works will revolve on discriminating different gestures to achieve full drumkit functionality, i.e. using positional information to play different sounds according to the drum or percussion instrument that the user wants to play.

Acknowledgments. This work has been funded by the Ministerio de Economía y Competitividad of the Spanish Government under Project No. TIN2010-21089-C03-02 and Project No. IPT-2011-0885-430000 and by the Junta de Andalucía under Project No. P11-TIC-7154. The work has been done at Universidad de Málaga. Campus de Excelencia Internacional Andalucía Tech.

References

1. Antle, A., Droumeva, M., Corness, G.: Playing with the sound maker: do embodied metaphors help children learn? In: Proceedings of the 7th international conference on Interaction design and children. pp. 178–185. ACM (2008)
2. Bakker, S., van den Hoven, E., Antle, A.: Moso tangibles: evaluating embodied learning. In: Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction. pp. 85–92. ACM (2011)
3. Barbancho, I., Rosa-Pujazón, A., Tardón, L.J., Barbancho, A.M.: Human–computer interaction and music. In: Sound-Perception-Performance, pp. 367–389. Springer (2013)

4. Bishop, C., et al.: Pattern recognition and machine learning, vol. 4. springer New York (2006)
5. Castellano, G., Bresin, R., Camurri, A., Volpe, G.: Expressive control of music and visual media by full-body movement. In: Proceedings of the 7th international conference on New interfaces for musical expression. pp. 390–391. ACM (2007)
6. De Dreu, M., Van der Wilk, A., Poppe, E., Kwakkel, G., Van Wegen, E.: Rehabilitation, exercise therapy and music in patients with parkinson's disease: a meta-analysis of the effects of music-based movement therapy on walking ability, balance and quality of life. Parkinsonism & Related Disorders 18, S114–S119 (2012)
7. Essl, G., Rohs, M.: Interactivity for mobile music-making. Organised Sound 14(02), 197–207 (2009)
8. Gower, L., McDowall, J.: Interactive music video games and children's musical development. British Journal of Music Education 29(01), 91–105 (2012)
9. Halpern, M., Tholander, J., Evjen, M., Davis, S., Ehrlich, A., Schustak, K., Baumer, E., Gay, G.: Moboogie: creative expression through whole body musical interaction. In: Proceedings of the 2011 annual conference on Human factors in computing systems. pp. 557–560. ACM (2011)
10. Holland, S., Bouwer, A., Dalgelish, M., Hurtig, T.: Feeling the beat where it counts: fostering multi-limb rhythm skills with the haptic drum kit. In: Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction. pp. 21–28. ACM (2010)
11. Höofer, A., Hadjakos, A., Mühlhäuser, M.: Gyroscope-Based Conducting Gesture Recognition. In: Proceedings of the International Conference on New Interfaces for Musical Expression. pp. 175–176 (2009), `http://www.nime.org/proceedings/2009/nime2009_175.pdf`
12. Jordà, S.: The reactable: tangible and tabletop music performance. In: Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems. pp. 2989–2994. ACM (2010)
13. Junker, G.: Pro OGRE 3D programming. Apress (2006)
14. Khoo, E., Merritt, T., Fei, V., Liu, W., Rahaman, H., Prasad, J., Marsh, T.: Body music: physical exploration of music theory. In: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games. pp. 35–42 (2008)
15. Levin, G., Lieberman, Z.: In-situ speech visualization in real-time interactive installation and performance. In: Non-Photorealistic Animation and Rendering: Proceedings of the 3 rd international symposium on Non-photorealistic animation and rendering. vol. 7, pp. 7–14 (2004)
16. Livingston, M.A., Sebastian, J., Ai, Z., Decker, J.W.: Performance measurements for the microsoft kinect skeleton. In: Virtual Reality Workshops (VR), 2012 IEEE. pp. 119–120. IEEE (2012)
17. Mandanici, M., Sapir, S.: Disembodied voices: A kinect virtual choir conductor. http://www.smcnetwork.org/system/files/smc2012-174.pdf, last retrieved 20/09/2012 (2012)
18. Morita, H., Hashimoto, S., Ohteru, S.: A computer music system that follows a human conductor. Computer 24(7), 44–53 (1991)
19. Ng, K.: Music via motion: transdomain mapping of motion and sound for interactive performances. Proceedings of the IEEE 92(4), 645–655 (2004)
20. Odowichuk, G., Trail, S., Driessen, P., Nie, W., Page, W.: Sensor fusion: Towards a fully expressive 3d music control interface. In: Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on. pp. 836–841. IEEE (2011)

21. Parton, K., Edwards, G.: Features of conductor gesture: Towards a framework for analysis within interaction. In: The Second International Conference on Music Communication Science, 3-4 December 2009, Sydney, Australia (2009)
22. Qin, Y.: A study of wii/kinect controller as musical controllers. http://www.music.mcgill.ca/ ying/McGill/MUMT620/Wii-Kinect.pdf, last retrieved 20/09/2012
23. Shreiner, D.: OpenGL reference manual: The official reference document to OpenGL, version 1.2. Addison-Wesley Longman Publishing Co., Inc. (1999)
24. Stierman, C.: KiNotes: Mapping musical scales to gestures in a Kinect-based interface for musican expression. Ph.D. thesis, MSc Thesis University of Amsterdam (2012)
25. Todoroff, T., Leroy, J., Picard-Limpens, C.: Orchestra: Wireless sensor system for augmented performances & fusion with kinect. QPSR of the numediart research program 4(2) (2011)
26. Trail, S., Dean, M., Tavares, T., Odowichuk, G., Driessen, P., Schloss, W., Tzanetakis, G.: Non-invasive sensing and gesture control for pitched percussion hyperinstruments using the kinect (2012)
27. Wang, C., Lai, A.: Development of a mobile rhythm learning system based on digital game-based learning companion. Edutainment Technologies. Educational Games and Virtual Reality/Augmented Reality Applications pp. 92–100 (2011)
28. Wiener, N.: Extrapolation, interpolation, and smoothing of stationary time series (1964)
29. Yoo, M.J., Beak, J.W., Lee, I.K.: Creating musical expression using kinect. Proc. New Interfaces for Musical Expression, Oslo, Norway (2011)

A Low-Cost VR System for Immersive FPS Games

José P. Molina[1], Arturo S. García[2], Jonatan Martínez[1], Pascual González[1]

[1] Laboratorio de Interacción con el Usuario e Ingeniería del Software (LoUISE)
Instituto de Investigación en Informática de Albacete (I3A)
Universidad de Castilla-La Mancha (UCLM)
Campus Universitario, Avda. España s/n, 02071 Albacete
{ JosePascual.Molina, Jonatan.Martinez, Pascual.Gonzalez }@uclm.es

[2] SymbiaIT, Albacete, Spain
arturo@symbiait.com

Abstract. This paper represents a continuation of our efforts to bring virtual reality to video games, specifically to first-person shooting games. Advocating for the need to decouple the weapon from the view, in a previous experiment we demonstrated with a precise -but expensive- optical tracking system that it was possible to disregard the position of the head and the hand of the user, and that only their orientations need to be captured. Based on the findings of that experiment, we now present a prototype that uses inertial technology available to everyone, two Wiimotes with Motion Plus extensions, with which we intend to show that it is also possible to transfer the results from laboratory to home at an affordable price. Preliminary tests with the prototype are satisfactory and open a promising future for transforming the way in which players are immersed in a FPS game.

Keywords. Video games, first-person shooters, virtual reality, human-computer interaction.

1 Introduction

The development of the Oculus Rift [1] head mounted display (HMD) has attracted the interest of players, developers and researchers. This HMD aims to offer professional HMD features to the videogame consumer, with wide field of view (FOV) and high resolution, such as those found in research laboratories and leading companies, but at a much lower price, enough to be attractive to the videogame consumer market. In addition to the above-mentioned features, this device also includes a head tracker [2] that provides the user's head orientation with a faster response time than other previous devices. With this combination of display and head tracker, the developers intend that any player can dive into any 3D videogame in first person, especially first-person shooters (FPS), such as Quake [3], created by John Carmack, who has also publicly supported of the Oculus Rift project. To this end, it is not necessary for the game to have specific support for the display or the head tracker. Thus, on the one

hand, the display is connected to the computer like any other monitor, so it is only necessary to adjust the point of view FOV to match the HMD. And, on the other hand, the head tracker has only to emulate the mouse, that is, the operating system recognizes it as a regular pointing device, and the head rotations up, down, left and right are translated into mouse movements up, down, left and right.

The possibility of using a head tracker and a HMD in a 3D first person videogame is not new, as it is something that could be found in earlier devices. In fact, we can find previous examples, as the Victormaxx HMD [4] with mechanical tracking launched in the last decade of the last century; or the Trimersion HMD [5] in the first decade of the present century, which not only included tracking, but it also included a wireless communication. It can be said that these two products failed because of their poor display specifications compared to those offered by the monitors they had to compete against. The visual fatigue can be added to these drawbacks, which is usually produced by the prolonged use of a HMD. However, despite these inconveniences, these systems would have succeeded if the experience had been worthwhile to the player, who would forget about the bad resolution and eye strain until the end of the game. The biggest problem lies in the design of first person shooters, that in order to reduce the number of degrees of freedom of the character, and to make it more user-friendly, combine the view and the gun under the same control, typically the mouse, so that the user looks where he points his weapon, and shoots where he stares to. Certainly, the user would want to shoot towards the target in front of him more often than to any other direction, and this reduction the degrees of freedom makes it easier and more comfortable the task of shooting down the enemies. But when the user wears the HMD, and the head tracker takes control of the view and the weapon emulating the mouse, this task is no longer comfortable or easy, because although looking around by turning the head is more natural, aiming with it is not, which results not only in eye-strain but also in neck fatigue.

For this reason, in order to enjoy the virtual reality technology with our favourite first-person shooter, it is necessary to separate view and weapon, having each one its own controls. It seems clear that the head tracker should keep its original function, which is to capture the head orientation and map it to the player point of view in the game. Therefore, a different control for the weapon must be found. This control can be a joystick or a gamepad, as evaluated in [6], and it can even be shaped as a weapon, like the controller shipped with the Trimersion HMD. But beyond that, it is possible to provide more realism to the game if we can capture the position and orientation of both the user's head and hand (or the device held by the hand). This is the intention, for example, of the company VRcade [7], using the Oculus Rift as the HMD but adding an optical tracking system which captures in real time the position and orientation of both the HMD and a weapon-shaped device carried by the user, thanks to the reflective motion-capture balls mounted on them. And this is not new, since at the beginning of the 90s the company W industries, later renamed as Virtuality [8], installed entertainment systems worldwide in arcade rooms, which included a HMD and a joy-stick, tracking their position and orientation thanks to an electromagnetic tracking system. Games running on these vintage platforms already separated gun and sight, as for example Dactyl Nightmare [9]. The problem is that, although the prices

of these computers are now lower, they remain out of the reach of the average consumer, both in terms of money and space. For example, an OptiTrack [10] system costs in the order of thousands euros, and requires also a space of about 5x5 meters (25 m2) to create a capture zone of 3x3 meters (9 m2).

Beyond the money or space, the first thing that the videogame industry will be more interested in, is whether it is worth, that is, if the game experience perceived by the player can outweigh the cost. In this case, the next step is to see if it can be put into practice, from the point of view of both software and hardware and, focusing on the later, if the cost and space requirements can be reduced to a reasonable magnitude.

Therefore, in the case of software, it should not be too hard to separate the movements of the eye and the gun, possibly also adding some more items in the setup menu in order to establish the separated mapping of the sight and the weapon with their corresponding inputs. The problem, if anything, is in the operating system, which is only able to recognize one mouse, so while the support of the head tracker can be eased emulating the mouse axes, the weapon tracking requires a different approximation, for example emulating the axes of a joystick or gamepad. In fact, although this is not usually found in commercial first-person shooters, it is found in many other games, especially sea, land or aircraft simulation, where the user can steer the vehicle towards one direction while looking around, as in car racing, flight simulators or tank battles. In the latter case, for example, there are many degrees of freedom when commanding a tank, the hull can move in one direction while the turret rotates freely, the barrel can aim with a given elevation angle, and even the commander can see a different sight leaning out of the hatch.

Regarding hardware, in recent years we have seen new generations of game consoles that introduced novel devices that capture user movements, either of the hand or of the whole body. These systems are based on inertial technology –accelerometers, gyroscopes- such as the Wii and Wii U controls [11]; and optical technology, such as Sony Playstation Move [12] or Microsoft Kinect [13]. The same technologies have also been introduced in the PC videogames market, with joysticks and gamepads that include the same inertial technology, like the Cyberstik XP and G-Pro controllers, and even electromagnetic tracking systems, like the Razer Hydra [14]. The interesting thing about these systems is that they try to limit their space demands to the space available at home, and more importantly, the cost is really affordable to the player.

All in all, we postulate in this paper that there are benefits for the player, and that it is possible to transfer this technology to his home with an affordable cost and with acceptable space constraints. Thus, those benefits will be identified in the following sections, then a demonstration will follow of how it is possible to reduce the required space doing it without the position information, and finally a prototype will be described that takes this concept into practice. The paper ends with the conclusions that can be extracted from this work, and future work proposals.

2 The benefits of an immersive experience for the player

This section is a review of different studies comparing the most common way to play a first-person shooter (using a monitor, a keyboard and a mouse, and with the weapon coupled with the eye and directed by the mouse) to other alternatives based on virtual reality technologies. These technologies aim to increase the fidelity of the action while pursuing a more engaging and entertaining game for the user.

Thus, the authors of the experiment described in [15] created a first-person shooter in which users had to take down a certain number of ghosts. They had to do it twice, the first time using the classical input configuration (mouse and keyboard) and seated in front of the monitor, and the second time also seated but using a HMD and a data glove, with a sensor on the HMD to let the user look around and another on the wrist to let they aim. In this case, the image presented on screen was drawn taking the position and orientation of the sensor located on the helmet, while just the orientation was taken from the wrist sensor to move a cursor on the image. The virtual reality equipment consisted of a VFX3D helmet with a resolution of 263x480 pixels and 24° of horizontal FOV [16], a Pinchglove and a Flock of Birds electromagnetic tracking system. The findings of this experiment showed that, although 85% of participants preferred to aim with the mouse because it had a higher hit rate, the preference for separating the point of view from the gun were divided to 50%. Also, 60% of participants preferred the helmet to look for the ghosts, while 90% claimed it was more fun. Finally, everyone agreed that it was more immersive.

In [17] a first-person shooter was also evaluated comparing the classic set-up of monitor, keyboard and mouse to another immersive set-up, but this time the game involved two users and, in the immersive set-up, both of them played standing, not seated. To immerse the user in the game, the authors used a HMD and a wand, both with position and orientation sensors. Again, the authors did not use the gun position. First, the scene is rendered using the position of the helmet sensor and the orientation of the wand sensor, and then displayed as a thumbnail in the corner of the screen, what they called Gun Radar. The pixel right in the center of the Gun Radar is where bullets are shot to. Then, they take the depth value of this pixel and render the scene again from the position and orientation of the helmet sensor. The virtual reality equipment was composed of an InterSense IS900 hybrid tracking system (inertial-ultrasonic) capturing a space of 2x4 meter, using 4 sensors, 2 HMD and 2 wands. Although it is not explicitly stated, looking at the photos it can be said that that the HMD used is the GVD510 by Oriscape [18], with a resolution of 640x480 pixels and a diagonal FOV of 28º. The conclusions of this work are that users perceive that the wand is as precise as the mouse, but it is more fun, and it is also clear that the users preferred the immersive configuration.

For our part, we describe in [6] an experiment where users also tested a multiplayer first-person shooter, particularly one trying to reproduce Virtuality's Dactyl Nightmare game. This time, the users did not only use two configurations, but a range of combinations of output devices -monitor and HMD-, input devices -keyboard, mouse, gamepad, joystick, air joystick and head tracker-, and several ways to map these input devices to the degrees of freedom of the point of view and the weapon, ranging from

fully coupled to fully decoupled, evaluating possible intermediate combinations. A total amount of eight configurations were evaluated. The HMD was one PC/SVGA i-glasses with a resolution of 800x600 pixels and a diagonal FOV of 26º [19], and the head tracker was an InterSense Intertrax 2 with 3 degrees of freedom. The air joystick was a Cyberstik XP with an accelerometer capable of measuring 2 degrees of freedom of movement in the air (pitch and roll). The findings of the experiment were that the combination that provided greater sense of immersion, allowed to better direct and coordinate the point of view and the weapon, and obtained the highest rating from the users that tested them was the one formed by the HMD, the head tracker and the gamepad, being the gamepad used to control both the gun and the user movement.

More recently, in [20], four different configurations are also compared for the same first-person-shooter. These configurations are the result of combining two types of displays, labeled as low and high fidelity (LD and HD) and that correspond to a projection screen and a six-sided CAVE, and two types of input, which are also labeled as low and high fidelity (HI and LI) and correspond to the mouse and keyboard combination on one side, and 3D glasses and wand, on the other. In this case, the position of the gun was used to control the gun, so the high-fidelity input took both the position and orientation of the 3D glasses to draw the scene from the point of view of the user. In the same way, the position and orientation of the wand was also used to aim and fire the gun. On the other hand, the combination of low fidelity in display and input introduced several differences to the classic layout of a user sitting in front of the monitor playing with keyboard and mouse, as this time the display was a projection screen and the user was standing, with the keyboard and mouse on a pedestal. In order to avoid dizziness, the mouse was not mapped on the point of view, but it moved a crosshair across the screen like a cursor in a window desktop. The virtual reality system used was a six-sided CAVE system, 3x3x3 meters and a resolution of 1050x1050 pixels per screen, stereoscopic glasses, a wand, and a hybrid InterSense IS900 tracking system. The findings of this study show that the combinations LDLI and HDHI are the fastest because of the user familiarity with FPS games, in the case of LDLI, and greater similarity with reality, in the case of HDHI. LI provides better accuracy because the users need fewer bullets, as they first move the cursor and then shoot instead of sweeping the horizon with their bullets, although the difference (49% vs. 43% of HI) is not much. In contrast, LI receives more damage because the user usually remains more static. Finally, HD allows seeing enemies using peripheral vision and HI gets lower times, being the HDHI the combination that provides most presence, engagement and usability.

In view of the results of these studies, we can then conclude that, although the classic configuration of monitor, keyboard and mouse usually provide better results in terms of accuracy, immersive settings allow the users to easily find the enemy with their eyes, and are preferred by players, either because of their greater immersion, realism and, above all, fun. In fact, in the evaluation described in [15], the participants seemed to forget the effort required by the head and hand movements and the fatigue they produce. More surprisingly, in these studies, both the resolution and the FOV of the HMDs used were far from the characteristics of professional devices, which is

what the Oculus Rift aims to bring to the users. Therefore, just a better HMD could, in any case, improve the results outlined here.

3 Orientation-only control models for immersive FPS games

Almost all the authors of the evaluations reviewed in the previous section do not use the position that captures the sensor on the hand that aims, or on the devices that the user holds in his hand, using solely the orientation given by that sensor to allow the user to aim in a direction different from the view. Only in one of these works, the one by McMahan et al., that position is used, that is, the authors read the position and orientation of the head sensor to create the user point of view in the scene, and read the position and orientation of the wand that the user holds to place and aim the weapon in the scene. This last work is, therefore, the one that most faithfully reproduces the action of aiming and shooting in the real world.

However, none of these authors questioned whether it is really necessary or not, for the game's goal, to pick the position of the head, and the hand or the device held in hand, in other words, if it is possible to do without these positions and still have a game just as fun with only the orientation of the tracked body parts and devices. After all, the FPS games for PC and consoles have had great success so far using one single orientation, the same for the view and the weapon, and the devices that capture rotational data, such as the inertial ones, are much cheaper than the systems that capture position as well as orientation, either optical, electromagnetic or hybrid, and also do not require the room that these other systems require for running. Sure, doing without position would make the game less realistic, but it comes without saying that this is not the main goal. The specific goal of a FPS game is to kill all the enemies, and to do that we can not lose accuracy, and the common goal of every game is to have fun, and to have that we must avoid making weapon aiming so easy that it becomes boring.

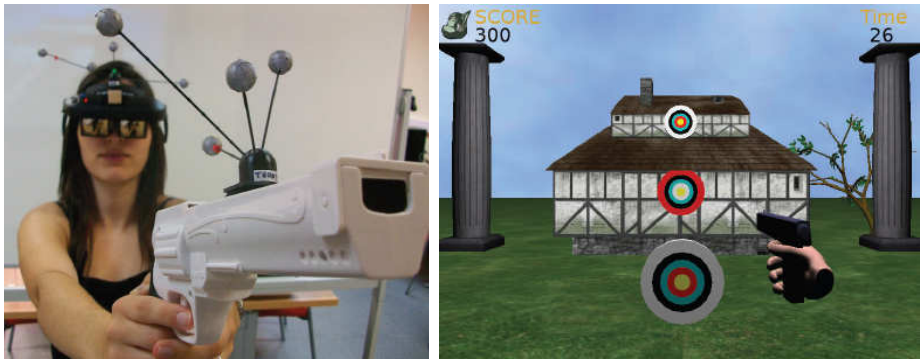


Fig. 1. Participant in tests (left), and screenshot of one of these tests (right)

With this idea in mind, in [21] we described the development of an immersive FPS game, using an i-glasses X2 as a virtual reality helmet, a Wiimote dressed as a revolver, and an iotracker optical tracking system [22] with two rigid bodies, one on the

i-glasses and the other one on top of the Wiimote (Fig. 1). With this game, an experiment was carried out to evaluate different control models, using the position and the orientation of both the user's head and the Wiimote in just one of them –the most realistic one-, and getting rid of the position in the rest of them, that is, capturing and using only the orientation of the head and the Wiimote.

Focusing on the models without position, these ones fill in the missing information using a preset position for the avatar's view –trying to macht the height of the user when he is standing- and for the elbow or the wrist of the avatar's arm that holds the weapon –relative to the view point, to the right or to the left of the trunk if the user holds the Wiimote revolver with one hand, or centred in front of the chest if the user yield it with two hands-. The user's head orientation is then mapped onto the avatar's view point, and the Wiimote orientation is mapped onto the avatar's elbow or wrist depending on the control model.

The evaluation proved that similar results to the position and orientation model were obtained with a model that mapped the gun tracked orientation to the avatar's elbow (Fig. 2), when holding the weapon with one hand, and even better results were obtained with the model that mapped the orientation on both elbows, when yielding the gun with two hands (Fig. 3). This confirmed the feasibility of replacing the position and orientation model with one that relies solely on the orientation, less realistic but, as was pointed by the participants in the questionnaires, just as fun.

Fig. 2. One-hand model, aiming with the elbow

Indeed, something that also became apparent during the tests was the willingness that users have to this new way of playing. After testing, the majority of users was surprised and praised the realism and the degree of immersion achieved. Some of them even asked if there was something similar for sale to enjoy at home. In short, users are very attracted by this technology. In this regard, the test that recreated an existing commercial game –Link's Crossbow Training- was the one that the users enjoyed most. The challenge of having a limited time and a score counter did certainly encourage their competitiveness. Almost everyone wanted to improve their initial score when they repeated the test, asking also for the final score of other participants to see how good they had played. This response from users made it clear the

possibility of bringing a system similar to the one proposed in this work to the video-game market.

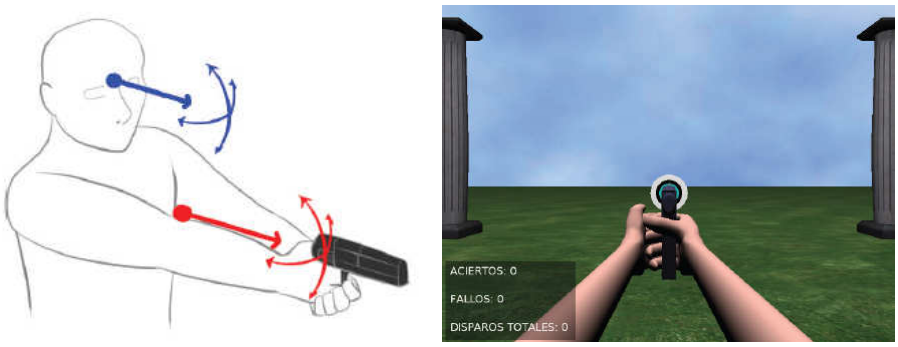


Fig. 3. Two-hand model, aiming with the elbows

4 Description of the prototype of the proposed VR system

One of the works that we proposed in [21], as a continuation to the conducted experiment, was the development of a prototype that could exploit the conclusions derived from that experiment and bring them to practice, that is, a low-cost virtual reality system that captures the user's head and hand orientation and, with that prototype, then repeat the evaluation to verify that the results can be translated from the iotracker used in that experiment to this other virtual reality system. Thus, this section describes the prototype that is proposed by our laboratory.

Fig. 4. Wiimotes with Motion Plus extension (left), and prototype in use (right)

The developed prototype consists of a HMD –in particular, one i-glasses X2- and two Wiimotes with the Motion Plus [23] extension, one of them used as a head tracker and the other one as a controller, the latter also with Nunchuk extension (Fig. 4). Other authors have proposed their own solutions, such as the VR Kit – HMD [24], which includes one Spacepoint Fusion to capture the head orientation, and a Gametrak to capture the positions of the head and hand, with the hands holding a Wii-

mote and a Nunchuk; however, the proposed solution only uses the joystick and buttons of the Wiimote and Nunchuk, and the author recognizes that it would require a second Spacepoint Fusion for capturing the controller orientation, which it is solved in our prototype with the Motion Plus. A similar proposal is the UCAVE [25], which uses the orientation sensor of an iPod to capture the hip orientation, and a Razer Hydra electromagnetic tracking system with the emitter in the hip and the sensors on head and hand, to capture the position and orientation of the head and hand in relation to the hip; however, the authors admit that the system suffers a noticeable latency when the hip movements are frequent, which can be expected to occur in an immersive FPS, and our prototype does not require capturing positions and thus we can free the user from carrying the Hydra emitter in his belt.

The idea of using two Wiimotes with Motion Plus was thought to be the best due to several reasons. Firstly, the Wiimote is a controller that we had used before [21], and it is easy to find software to get data from it, but the 3-axis accelerometer of the original controller allowed only capturing the tilt –roll and pitch angles-, not the rotation about the vertical axis –yaw angle-; however, the Motion Plus accessory added three gyroscopes that, together with the accelerometer, turn the Wiimote into a 3-DOF orientation sensor capable to capture roll, pitch and, yaw angles. Moreover, the set is affordable, around 45 euros the controller and 15 euros the accessory, at the moment of writing this.

In order to use the Wiimotes, it was necessary to solve two problems: how to poll data from a Wiimote, and how to input that data to the FPS game. As said before, there are software libraries that makes it simple to read data from a Wiimote, however only few of them support the Motion Plus accessory, and those that support it only return raw data, not the angles we want. The WiiYourself! library [26], which was already used in [21], was found to be the most suitable one for our purposes, and this way we developed on top of it several functions to calibrate the data sent by the Motion Plus –rotation speed, given in degrees/second-, to combine the data with that obtained from the Wiimote accelerometer -sensor fusion-, and finally to calculate the desired roll, pitch, and yaw angles (Fig. 5). Important to note is that these angles are not so easy to calculate because we are not interested in absolute, independent angles, but instead we want to find the relative, dependent Euler angles that can be composed, in a given sequence, in a single transformation matrix.

The second problem that we needed to solve was how to deliver the Wiimote data to the FPS game. One way to do this would be to emulate other devices, such as the mouse or the joystick. However, we would lose one degree of freedom if we project the 3-DOF of the orientation sensor onto the 2-DOF of the mouse. Besides, given that the goal of the prototype was to serve as a technology demonstrator and to repeat the previous evaluation with it, it was thought that the best option was to use the VRPN library [27], already used in that experiment to deliver to the FPS game the data generated by the iotracker system. The fact is that with VRPN it is easy to change the system that produces the data –the VRPN server, the tracking system-, without changing the code of the application that consumes that data –the VRPN client, the FPS game-, because the data is packed in a standard format and delivered by network. Thus, we programmed a new VRPN server that, built upon the WiiYourself! library,

supports several Wiimotes with Motion Plus extension, is able to calculate the roll, pitch, and yaw angles, and sends the data through the network to the VRPN clients as separate values (Analog server, in terms of the VRPN library), or composed in a single quaternion (Tracker server), apart from other data of interest, such as the state of the buttons (Button server).

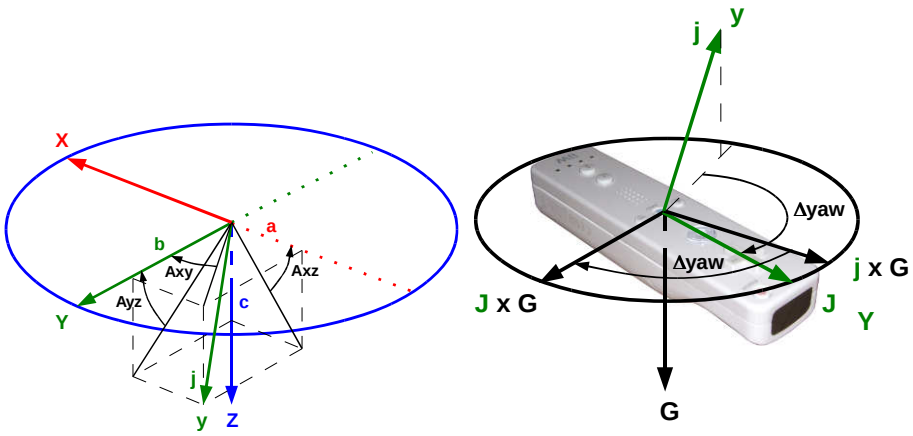


Fig. 5. To obtain the yaw angle, the gyro angles (Axy, Axz, Ayz) are first calculated from the gyro speeds for a lapse of time, then they are used to find the direction of the Wiimote longitudinal axis at the beginning of that lapse, y-axis, relative to the device XYZ local system at the end of it (left pic.), and finally the yaw is the angle between the projections of the axis before (y-axis) and after (Y-axis) onto a plane perpendicular to the gravity vector, **G** (right pic.)

The preliminary tests with the prototype revealed a satisfactory performance. As the system is based on inertial technology there are, in particular, two possible problems that deserve our attention: latency and drift. The latency is, in our system, the sum of the time to read the data from the Wiimote, to process it, to send it over the network, and to use it in the application. From these times, we can disregard the network transmission if, as in our case, both the server and the client run in the same computer. The processing time is our highest concern, because filtering and combining data from the acceleration and gyro sensors to get a stable, yet usable signal can increase latency in a significant way; in our server, however, the computation is fast and the filtering is reduced to a minimum, nevertheless obtaining a signal comfortable for the user.

With regards to the drift, this is the result of the accumulation of errors, in particular in the rotation about the vertical axis (yaw angle), given solely by the gyroscopes of the Motion Plus, as the errors in tilt (roll and pitch angles) are quickly and easily removed with the gravity vector that is obtained by the acceleration sensor of the Wiimote. The drift can be unnoticeable if the user only makes use of one single orientation sensor, as in the VR Kit, but it can become visible to the user with two sensors if the drift is different in each of these devices. Our first tests with the prototype, however, show promising results, and as a consequence we plan to start soon its formal evaluation.

5 Conclusions and future work

Although there have been, and there are, many proposals and attempts to bring virtual reality to video games, one that is receiving more publicity these days is the Oculus Rift helmet, especially oriented to the genre of first-person shooters. With its wide FOV and its head tracker it allows the player look and aim around. However, to avoid repeating past failures, it is imperative to separate view and weapon, tracking independently the head and hand of the user. In [21], we show that it is not necessary to capture the position, just the orientation, and in this paper we have presented and described a prototype that, based on two Wiimotes with Motion Plus, also shows how this result can be brought to the player at an affordable price.

The most immediate continuation of this work is the repetition of the previous experiment, this time with the developed prototype, although we would not need to repeat it completely, only with the control models that proved to be more valuable. Afterwards, it would be interesting to extend the experiment by allowing the user to move around the game scenario, possibly using the joystick on the Nunchuk, to evaluate whether this variable has any impact on the results so far obtained. As for the VRPN server, its code will be released to allow others to make use of it. To add it to the VRPN library distribution, however, it should be based on the Wiiuse library used in the server already included in the current VRPN release, but Wiiuse does not yet support the Motion Plus extension. In fact, the WiiYourself! library only supports Motion Plus as an extension to the original Wiimote, not the new Wiimote controller with Motion Plus inside, another issue that could be addressed in following versions.

Acknowledgements

This work has been supported by the project TIN2012-34003, funded by the Spanish Ministry of Science and Innovation.

References

1. Oculus VR, Inc.: Oculus Rift Headset. Accessed May 2013, URL: http://www.oculusvr.com/
2. Oculus VR, Inc.: Blog: Building a Sensor for Low Latency VR. January 2013, URL: http://www.oculusvr.com/blog/building-a-sensor-for-low-latency-vr/
3. Id Software: Quake. Accessed May 2013, URL: http://www.idsoftware.com/games/quake/
4. Gradecki, J.: The Virtual Reality Construction Kit. John Wiley (1994)
5. Kuntz, S.: The Story behind the Trimersion HMD. February 2007, URL: http://cb.nowan.net/blog/2007/02/09/the-story-behind-the-trimersion-hmd/
6. Delgado, F., García, A.S., Molina, J.P., Martínez, D., González, P.: Acercando la Tecnología de Realidad Virtual a los Videojuegos de Acción en Primera Persona. In: II Jornadas de Realidad Virtual y Entornos Virtuales, JOREVIR, Albacete, Spain (2008)
7. Brown, P.: VRcade Sets its Sights on the next Leap in Immersive Gaming. May 2013, URL: http://www.gamespot.com/features/vrcade-sets-its-sights-on-the-next-leap-in-immersive-gaming-6408035/

8. Cotton, B., Oliver, R.: Understanding Hypermedia: From Multimedia to Virtual Reality. Phaidon Press Ltd. (1992)
9. Arcade History: Dactyl Nightmare (1991). Accessed May 2013, URL: http://www.arcade-history.com/?n=dactyl-nightmare&page=detail&id=12493
10. NaturalPoint, Inc.: OptiTrack Motion Capture. Accessed May 2013, URL: http://www.naturalpoint.com/optitrack/
11. Lee, J.C.: Hacking the Nintendo Wii Remote. In: IEEE Pervasive Computing, 7(3), pp. 39-45 (2008)
12. Humphries, M.: Review: PlayStation Move, Geek.com. September 2010, URL: http://www.geek.com/articles/games/review-playstation-move-20100917/
13. Zhang, Z. Y.: Microsoft Kinect Sensor and Its Effect. In: IEEE Multimedia, 19(2). pp. 4-10 (2012)
14. Razer: Hydra. Accessed May 2013, URL: http://www.razerzone.com/es-es/gaming-controllers/razer-hydra
15. Torchelsen, R.P., Slomp, M., Spritzer, A., Nedel, L.P.: A Point-and-Shoot Technique for Immersive 3D Virtual Environments. In: Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital, SBGames, Brazil (2007)
16. Bungert, C.: Unofficial VFX3D page, Accessed May 2013, URL: http://www.stereo3d.com/vfx3d.htm
17. Zhou, Z.Y., Tedjokusumo, J., Winkler, S., Ni, B.: User Studies of a Multiplayer First Person Shooting Game with Tangible and Physical Interaction. In: Virtual Reality, Lecture Notes in Computer Science Volume 4563, pp. 738-747 (2007)
18. Oriscape: GVD510-3D Head Mounted Display. Accessed May 2013, URL: http://www.oriscape.com.cn/en/products.asp?sortid=251&id=118
19. I-O Display Systems: i-glasses. Accessed May 2013, URL: http://www.i-glassesstore.com/
20. McMahan, R.P., Bowman, D.A., Zielinski, D.J., Brady, R.B.: Evaluating Display Fidelity and Interaction Fidelity in a Virtual Reality Game. In: IEEE Transactions on Visualization and Computer Graphics, 19(4), IEEE Computer Society (2012)
21. Olivas, A., Molina, J.P., Martínez, J., González, P., Jiménez, A.S., Martínez, D.: Proposal and Evaluation of Models with and without Position for Immersive FPS Games. In: 13th International Conference on Interacción Persona-Ordenador, Elche, Spain (2012)
22. Pintaric T., Kaufmann H.: Affordable Infrared-Optical Pose-Tracking for Virtual and Augmented Reality. In: Trends and Issues in Tracking for Virtual Environments Workshop, IEEE VR 2007, Charlotte, NC, USA, pp. 44-51 (2007)
23. WiiBrew: Wii Motion Plus. Accessed May 2013, URL: http://wiibrew.org/wiki/Wiimote/Extension_Controllers/Wii_Motion_Plus
24. Kuntz, S.: VR Kit – HMD : Portable VR. June 2010, URL: http://cb.nowan.net/blog/2010/06/30/vr-kit-hmd-portable-vr/
25. Basu, A., Saupe, C., Refour, E., Raij, A., Johnsen, K.: Immersive 3DUI on one Dollar a Day. In: IEEE Symposium on 3D User Interfaces, 3DUI, pp. 97-100 (2012)
26. Gl·tter: WiiYourself! Native C++ Wiimote Library. Accessed May 2013, URL: http://wiiyourself.gl.tter.org/
27. Taylor II, R.M., Hudson, T.C., Seeger, A., Weber, H., Juliano, J., Helser, A.T.: VRPN: A Device-Independent, Network-Transparent VR Peripheral System. In: ACM Symposium on Virtual Reality Software & Technology, VRST, Banff Centre, Canada (2001)

UHotDraw: a GUI Framework to Simplify Draw Application Development in Unity 3D *

Ismael Sagredo-Olivenza, Gonzalo Flórez-Puga,
Marco Antonio Gómez-Martín and Pedro A. González-Calero

Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
email: isagredo@ucm.es, {gflorez,marcoa,pedro}@fdi.ucm.es

Abstract. Unity 3D is a widely used middleware for game development. In addition to the core technology, a number of extensions or plug-ins have been created by third-party developers to provide additional functionality. Nevertheless, the support Unity 3D provides for building GUI for such plug-ins is at a very low level, and for that reason we have designed and built UHotDraw, a framework in C# that simplify the development of GUI and draw applications in Unity 3D.

1 Introduction

The Computer revolution has brought mankind a plethora of tools that ease tasks that in the pre-digital era were made by hand. Nowadays, those tedious tasks that required several hours of hard work are done by machines driven by pieces of software in such a way that are able to complete the same job in less time and with minimum human supervision.

Computer engineers themselves have profited of this process and we have developed a wide range of tools whose unique goal is to ease the task of building other software. In the list of technologies and tools, we can include from high-level languages to those of the model-driven architecture (MDA) movement, by way of code-generator tools, version control systems, and so on.

The Game development area has not been immune to this tendency. In a typical game studio we find professionals of different areas such as game designers, programmers and artists (not to mention those roles not related to the development itself such as marketing). Traditionally, programmers have had two different responsibilities: in one hand they have to provide tools to the rest of the team (being the level editor, the main one); in the other hand, they develop the code for the game itself.

Over the last decade, however, a different approach has arisen, allowing programmers to be partly relieved of those responsibilities. They are known as *game engines*: a piece of software that integrates both development tools and game code. The entry point to the game engine is the level editor, where designers

* Supported by the Spanish Ministry of Science and Education (TIN2009-13692-C03-03)

build the different scenes that conforms the game. The level editor itself is usually in charge of creating the game executable for the target platform which usually incorporates levels, assets created by artists, game engine code and additional code created by the studio developers, including the specific behaviours that their game needs. The significant cost savings in development time and team size compensate for the cost of licensing the game engine.

According to a survey done to game developers by Gamasutra[1], a portal specialized on games, the two more valuable aspects of a game engine are the *rapid development time* and its *flexibility and easy extendability*. Regarding this last point, it involves two different aspects: how easy is to extend the game engine creating new behaviours and code to be added to *the final game* and the ability to extends the *tools themselves* incorporating functionality that is used by designers and artists at development time but that are not seen by final users.

One of the most used game engines is Unity3D[2]. According to the survey mentioned before, up to 53.1% of independent game studios are using it, mainly for game development over mobile platforms based on both Android and IOS.

Though those numbers support its quality to a large extent, we have identified a weak point that complicates the extensibility of the editor itself. As we will describe in the next Section, Unity neglects the way programmers may create Graphical User Interfaces (GUIs) and incorporate them into the editor. In fact, it does not even provide facilities in order to create GUIs for the game itself. Luckily, different libraries have appeared for building in-game GUIs (mainly menus and HUDs), but they cannot be used to add windows to the Unity editor itself.

This paper presents UHotDraw, a framework developed by the authors that ease the task of building GUIs to extend Unity editor. Our main motivation was the creation of a visual editor of non-player character (NPC) behaviours using mainly behaviour trees [6,5]. Therefore, UHotDraw incorporates not only the typical collection of widgets available in other general purpose GUI frameworks, but also an extended support for trees.

2 Unity 3D

Unity 3D is a relative modern game engine that has become popular in the last few years. It was created by the Danish company *Unity Technologies*. The philosophy behind this engine is to "democratize the game develop" trying to make it as much accessible as possible. This engine is widely used in the game industry, especially among the independent game studios, because it is powerful, easy, multiplatform and the license is less expensive than other engines.

Unity can be extended using scripting languages. Its extensibility makes possible the existence of a digital store that is smoothly integrated within the Unity Editor and allows users to buy extensions, meshes, textures, scripts and other

[1] `http://www.gamasutra.com/view/news/169846/Mobile_game_developer_survey_leans_heavily_toward_iOS_Unity.php#.UFze0I1lTAE`

[2] `http://unity3d.com/`

assets that are easily incorporated to the projects. Moreover, the companies that develop tools for Unity can share/sell their products through this store.

Regarding this extensibility, developers may create scripts using three different programming languages: C#, Javascript and Boo (a Python inspired language). Using them programmers may add *components* [8,7,1,3] to the project that are later assigned to game entities. Additionally, they may create utility classes that are used by those components during the game.

Unity also allows to extend the Unity Editor itself. The process consists on creating scripts and adding them to a special folder into the game project. The game engine assumes that those classes are meant to be used at development time and are not included in the final compilation of the game. In fact, the API provided by those classes is different from the one available for scripts created for the game (it has, nevertheless, some common functionality). One part of such API is related to the creation of windows and widgets into the editor allowing users to create their own tools and integrate them into the Unity interface.

Unfortunately, the API provided by Unity to create GUIs is far from perfect. As we will describe shortly, it is tedious, prone to errors and not very object oriented. Having as motivation the integration into Unity of a behaviour editor, this forces us to create a framework called UHotDraw that facilitates the creation of complex GUIs over the narrow Unity GUI API.

The main inconvenience of this GUI API is the lack of separation between the construction of widgets and the management of the events they fire. Unity does not provide any class that represents a widget like a button or label. Therefore, it is not possible to store a reference to them once created to change their state. The process, from the developer point of view, is quite different from the one used by frameworks like Swing [4] or Windows Forms[3].

To get a better idea of the implications, let us see a small example: imagine we want to create a minimal window with three buttons and a label. When pressing each button the label should change its alignment according to the button pressed (left, center or right).

In Unity (figure 1a) the construction of such window starts with the addition of a new class that inherits from the `EditorWindow` and the creation of just one method, `OnGUI`, that is called whenever there is any event on it. This method is responsible of both creating the buttons and label and processing the events. This is possible because the same method that creates/paints a widget like the button returns if that button was clicked last frame (if it existed). Managing the location of the widget is done using absolute coordinates; there is not any kind of layout manager that dynamically place the widgets depending on the available size.

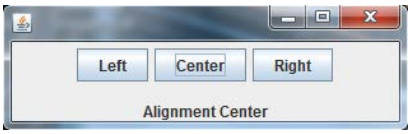
The code needed to create the windows described above is:

```
public class AlignmentExample : EditorWindow {

   public void AlignmentExample() {
      _labelPos = new Rect(25, 25, 100, 20);
```

[3] `http://msdn.microsoft.com/es-es/library/dd30h2yb.aspx`

(a) Window example in Unity



(b) Window example in Swing

Fig. 1: Example window

```
      _labelTxt = "Align Left";
    }

   void OnGUI(){
      GUI.Label(_labelPos,_labelTxt);

      createButtonAndProcessEvents(25, 25, "Left", 25);
      createButtonAndProcessEvents(150, 25, "Center", 150);
      createButtonAndProcessEvents(275, 25, "Right", 275);
   }

   void createButtonAndProcessEvents(
                        int posX, int posY,
                        String alignment, int labelPosX) {

      bool pressed = GUI.Button(
                             new Rect(posX, posY, 100, 20),
                             alignment);

      if (pressed) {
         _labelPos.x = labelPosX;
         _labelTxt = "Align " + alignment;
      }
   }
   private Rect _labelPos;
   private string _labelTxt;
}
```

As the code reveals, there are no objects representing the buttons nor the label. On the one hand we need to "create" all the widgets over and over again to keep them on the screen. On the other hand we have to process the button-pressed event on the same section of code where we create it. Moreover, as Unity does not track the widgets that we created last frame, we are responsible of maintaining their state (position, caption). In the example above, we need to store the text and position where we place the label. Just to compare this tedious approach with other GUI frameworks, let us see how we can be develop a similar functionality using the Java Swing API (figure 1b):

```
public class AlignmentWindow extends JFrame {
```

```
  public AlignmentWindow() {
    setSize(320, 100);
    JPanel buttons = new JPanel();
    this.add(buttons);
    buttons.add(new AlignmentButton("Left", LEFT));
    buttons.add(new AlignmentButton("Center", CENTER));
    buttons.add(new AlignmentButton("Right", RIGHT));
    label = new JLabel("Alignment Left");
    this.add(label, BorderLayout.SOUTH);
  }

  class AlignmentButton extends JButton
                        implements ActionListener {
    public AlignmentButton(String text, int alignment) {
      super(text);
      labelText = text; labelAlignment = alignment;
      addActionListener(this);
     }
     public void actionPerformed(ActionEvent e) {
      label.setText("Alignment " + labelText);
      label.setHorizontalAlignment(labelAlignment);
     }
     String labelText;
     int labelAlignment;
  }
  JLabel label;
}
```

Though the code is a bit larger, its design is clearer. Class constructor is responsible of creating the widgets. Once they are created, developer does not have to worry about tracking them for painting. Moreover, we are able to extend the `JButton` class to create a new kind of buttons that, when pressed, change the text and alignment of the label according to some parameters given to them during its construction (and stored in the button as attributes). This class design is more extensible, easy to use and fits well with the Model-View-Controller architectural pattern [2] (MVC) that is extensively used to create GUIs.

Having seen this small example, it is plain that Unity GUI support is poor and difficult to use for the creation of complex GUIs. It may be objected that not many people will try to extend the editor itself with their own functionality. However the lack of GUI support is also suffered by the in-game GUI, because the API is mainly the same. Luckily, different initiatives have appeared to help developers in the creation of these in-game GUIs. To mention just a few, Unity Store makes available popular libraries such us NGUI or UniGUI that joins a good object oriented design and more professional results. Unfortunately those libraries are based on capabilities such as render to texture that are not available in editor mode, and therefore they cannot be used to extend it.

As far as we know there is only one extension, called EWS GUI Editor [4], that allows to easily extend both editor and games. However, it only allows the creation of forms and rendering textures but not the complete set of features needed to build a plug-in like the one we had in mind. EWS GUI Editor is not a framework but rather a graphical editor for the Unity GUI and doesn't allow to create complex layouts like trees, graphs or hierarchical containers. This editor is designed to create simple widgets like boxes, buttons, text fields, textures, sliders, etc. Also, this tool is not extensible and is a simple wrapper of Unity GUI, and not provides a oriented object GUI framework.

In the figure 2 we can see a screenshot of EWS GUI Editor showing the type of graphical interfaces that this editor allows to build.

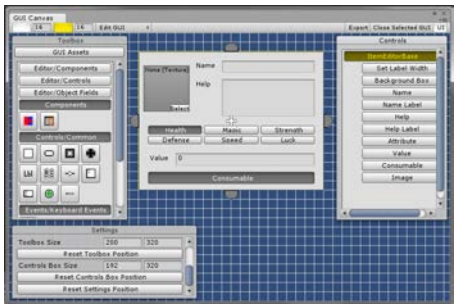


Fig. 2: EWS GUI Editor interface example

At this point, we were forced to develop UHotDraw, a framework that, using the limited Unity capabilities, eases the task of building a complete graphical editor and integrate it, not only into the editor but also into the game itself (if that were needed in the future). During the design process of the framework we analysed some GUI frameworks that are described in the next section and the source code of Strumpy Shader Editor. This editor is an extension of Unity that can be found in the Unity Store and presents the user some widgets similar to the ones we wanted for our behaviour editor. Unfortunately, we found that the implementation of the plug in was based on the `OnGUI` method without any kind of abstraction layer or framework, so we discarded it as a source of inspiration.

3 Other graphical editor frameworks

For the creation of our framework we have documented us and we analysed various graphical editor frameworks, including JHotDraw and GEF.

Both are two good examples of how a graphical editor framework should be structured. Furthermore, both use a lot of design patterns that we can reuse in our own framework for Unity.

[4] `https://www.assetstore.unity3d.com/#/content/8391`

3.1 JHotDraw

JHotDraw is a Java GUI framework for creating structured graphics. According to its own creators, Erich Gamma and Thomas Eggenschwiler, this framework is aimed to be a "design exercise" but also, it is already a quite powerful tool to be used in a lot of domains [5].

JHotDraw uses a Model-View-Controller paradigm which separates application logic from user interface dependencies. In this paradigm, the view is usually responsible for displaying the information and drawing the user interface. The controller handles the user input events and serves as an intermediary between the model and view and finally the model is a logic representation of the application. We can see a scheme the MVC design parading in the figure 3.

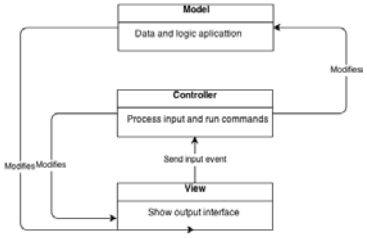


Fig. 3: Model view controller MVC

The basic components of JHotDraw's architecture are shown in the figure 4. The editor allows the creation of two basic figures: the graphical composite figures and Line Connections. Any other shape must inherit from these two basic shapes.

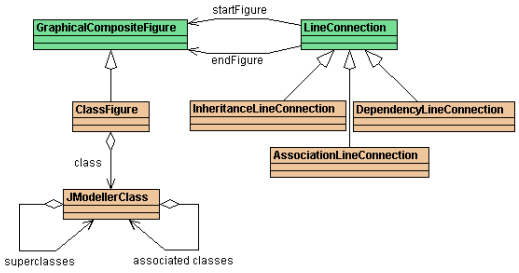


Fig. 4: Basic components of JHotDraw

Some design features of JHotDraw are:

[5] http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-jhotdraw.html

– Uses the Composite pattern to create shape containers.
– A Locator hierarchy that allows to choose the type of positioning among: relative, absolute, offset, etc.
– AbstractHandle for manipulate the graphics shapes allowing scale, rotate, set anchor points, etc.

JHotDraw implements a lot of design patterns that we can implements in other graphical editor frameworks. It is a really pedagogical tool, that we used as inspiration in many design decisions in our framework. For example, our framework using the MVC scheme, and implement a Composite pattern for the graphics containers. Also, we use a similar system of location types.

3.2 GEF

GEF (Graphical Editing Framework) is a set of Eclipse extensions that provides technology to create graphical editors. GEF also uses the MVC paradigm and is composed of three different tools: Draw2d, GEF (MVC) and Zest.

Draw2d is a layout and rendering toolkit building on top of Java SWT. All 2d figures in Draw2d are simple java objects that have no corresponding resource in the operating system.

GEF (MVC) adds editing behaviour on top of Java SWT (for example, it allows to create Trees, TreeItems, etc) and provides a rich set of tree-based elements and graphical editors for Eclipse.

Zest is a visualization toolkit that provides a set of layouting algorithms many interesting to apply in several forms of drawing.

In particular, our framework is inspired by GEF (MVC) and its fantastic trees and also in Zest tree algorithms. In Gef is easy to construct a `TreeView` inheriting from `AbstractEditPartViewer` class and implement corresponding listener that `TreeView` need.

4 Our Framework

Generally speaking, a framework is a piece of software that provides generic functionality that may be changed and extended by developers adding application-specific code. By using frameworks, developers can benefit from using already tested code that provides with a basic skeleton of an application with well defined *hot spots* or *hooks*. These *hooks* are the extensibility points were programmers may add their own code to build their applications. In order for a framework to be useful, it must be adaptable to the needs of the problem domain which, usually, is not known when the framework is designed.

Ideally, the framework must also hide the peculiarities of the target platform with an abstraction layer that provides an easy-to-use API.

As it has been explained before, the main peculiarity of our platform is the existence of just one method as the entry point to the GUI system, the `OnGUI` method from `EditorWindow`. Using UHotDraw programmers do not have to

worry about it but just to extend a more versatile class called `GenericWindow`. This class manages a list of the widgets being added to it and implements the method `OnGUI`, that calls Unity for having them painted and find out whether user events have been fired. When one of such events arise, the method itself invokes other internal methods that will end up calling application-specific code.

The basic element that can be added to the window is represented by the abstract class `AGraphicObject` (see a partial view of the class diagram of our framework in figure 5). Different widgets and shapes are implemented as derived classes from it, such as `Label` or `AGraphicButton`. They contain attributes with their state (size, caption) and internal code that is called from the window the widget belongs to.

UHotDraw also supports the creation of widgets hierarchies using containers. In particular, the class `AGraphicContainer` is just a container of other `AGraphicObject` (using the composite design pattern [2]). Nowadays, UHotDraw has two containers. Both of them, when added to the parent container, are placed on a specific location (whose coordinates may be given as absolute or relative coordinates) and given a size. The main difference between both of them is their behaviour when other widgets are added to them and their extent exceeded the limits of the container itself. On the one hand, `ClippingContainer` just *clips* (or hide) those parts of the widgets that go beyond its limits. On the other hand, `ScrollingContainer` presents both horizontal or vertical scrollbars to let the user select the specific area drawn on it.

Additionally, UHotDraw promotes the use of the MVC pattern in the applications using it. Without taking into account the low level of abstraction (where the caption of a button could be considered the *model* of that button) and staying at the level of the developer that uses UHotDraw, most of the classes mentioned above are considered part of the *view* because they are responsible of rendering themselves. They are also in charge of managing the UI events and invoking external code that it is placed on the *controller* part of the MVC. This invocation may be done using events/delegates (a high-level construction of several high level languages such as C#), using the traditional observer pattern (in a way similar to the one that appeared above in the Swing example) or thanks to the overriding of a method in the base class (see the example below). Finally, the *controller* will call other application-specific code that represents the domain model being developed.

When creating the framework design, we had always the extensibility in our sights. We have set some *Hot Spots* to extend its default functionality. The framework user can create other types of `AGraphicsContainer` with special functionality, composing them with other containers or graphic objects. The user can create new `Shapes` and line connectors as well as new locations and `ASizeComps` to adapt the location and size to their needs.

Additionally, UHotDraw provides a number of features that, while not strictly related to the graphic GUI, are often widely used in most graphical editors. One of these utilities is the undo/redo capability. The framework includes an action manager (`ActionMgr`) that is responsible of issuing the different commands that

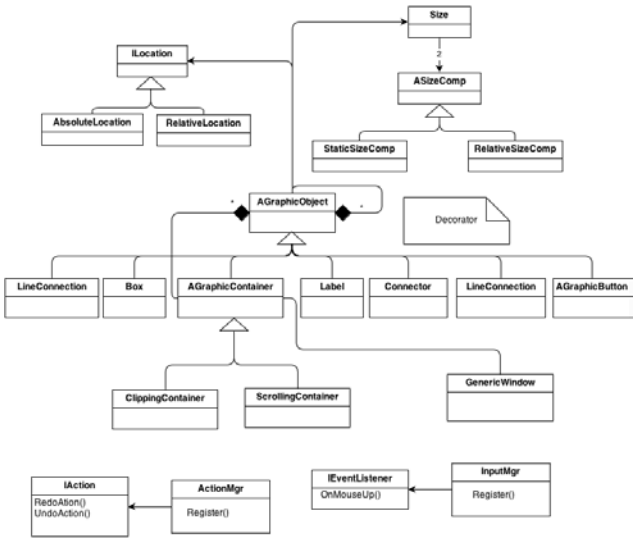


Fig. 5: Partial class diagram of UHotDraw

affect the model. Each command has to implement the interface `IAction`, that contains the methods that have to be implemented to execute and undo each particular action. By invoking these methods, the manager can undo and redo chains of actions.

Other interesting feature is the set of mouse and keyboard events in the `InputMgr`. If the user application needs to react to these events, it can implement the `IEventListener` interface and register then in the `InputMgr`.

Lastly, our framework is developed on top of Unity GUI, meaning that almost all elements can be displayed as game GUI. This unifies the way to build GUIs regardless of whether we made an editor GUI or game GUI and help to simplify the cumbersome GUI provided by Unity.

The following listing shows the implementation of the example from Section 2 using the classes in UHotDraw. The main window is a class that inherits from `GenericWindow`. The `Init` method is responsible of creating (and adding) the containers and widgets to the window. It does that via the `AddChild` method implemented by UHotDraw. Though the details have been omitted on the code shown below, the method instantiates the label that will contain a text and it will change the text alignment in the tree buttons. As the swing implementation, the buttons are implemented using a new class that inherits from `AGraphicButton`. Now, however, instead of having the class to implement an interface we have chosen override the method `OnButtonPressed` and add there the event management.

```
public class AlignmentWindow : GenericWindow  {
```

```
  public void ArticleTest() {
    Init();
  }

  protected override void Init(){
    AbsoluteLocator position = ...;
    StaticSizeComp width = ..., height = ...;
    _label = new Label(position, width, height, "Align Left");
    AddChild(_label);

    position = new AbsoluteLocator(new Vector2(25, 50));
    width = ...; height = ...;
    AddChild(new AlignButton("Left", position, width, height));

    position = new AbsoluteLocator(new Vector2(150,50));
    AddChild(new AlignButton("Center", position, width, height));

    position = new AbsoluteLocator(new Vector2(275,50));
    AddChild(new AlignButton("Right", position, width, height));
  }
  protected static Label _label;

  public class AlignButton : AGraphicButton {
    // [Default constructor ommited]

    protected override void OnButtonPressed() {
      Vector2 pos = new Vector2(this.GetLocator().GetPosition());
      pos.y += 100;
      _label.SetText("Align " + this.getCaption());
      _label.GetLocator().SetPosition(pos);
    }
  }
}
```

A more complete example of using the framework can be seen in Figure 6 where we have implemented two different drawing areas. On the left side, we can see a container where we draw graphics elements. In the right area, is intended to draw buttons and other controls to implement different tools. The buttons extend the class `AGraphicButton`, and override the `OnClick` method. The canvas is a class that extends `ScrollContainer` to implement a `TreeView` container. The class `TreeNode` is a `ClippingContainer` that contains various graphics shapes. The lines inherit from the class `LineConnection` with a special location that needs the begin and end position to be drawn.

5 Conclusions

Unity is a game engine widely used by developers, but with a poor API for building GUIs that it is hardly extensible and maintainable. A detailed analysis of

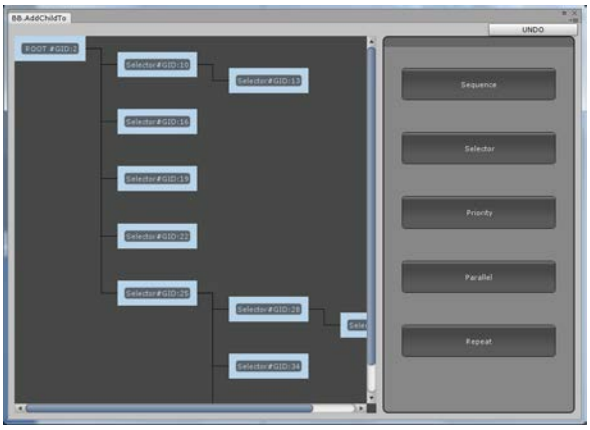


Fig. 6: Framework example

available plug-ins in the Unity Store, where third-party plug-ins are distributed, led us to the conclusion that, although exists some support technology for building in-game GUIs, there are not many tools to help creating an editor extensions itself .

UHotDraw, the framework presented here, has been developed following well-know design principles for GUIs frameworks. In particular we take ideas from JHotDraw, Swing and Gef. Our framework facilitates the creation of GUIs and graphical editors in Unity, with a software design that promotes extensibility and maintainability, through the well known MVC pattern, that separates the underlying logic from the code of the event processing. Furthermore, as the framework is built over Unity GUI API, it cannot only be used to create plug-ins but also to develop in-game GUIs easily.

In the future we plan to use this framework to develop tools that facilitate the creation of intelligent characters for games, within the Unity editor.

References

1. W. Buchanan. *Game Programming Gems 5*, chapter A Generic Component Library. Charles River Media, 2005.
2. R. J. Enrich Gamma, Richard Helm, editor. *Desingn Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Professional Computing Series, 1994.
3. S. Garcés. *AI Game Programming Wisdom III*, chapter Flexible Object-Composition Architecture. Charles River Media, 2006.
4. P. V. Matthew Robinson, editor. *Swing, Second Edition*. Manning, 2003.
5. S. Rabin, editor. *AI Game Programming Wisdom 3*. Charles River Media, 2006.
6. S. Rabin, editor. *AI Game Programming Wisdom 4*. Charles River Media, 2008.
7. B. Rene. *Game Programming Gems 5*, chapter Component Based Object Management. Charles River Media, 2005.
8. M. West. Evolve your hiearchy. *Game Developer*, 13(3):51–54, Mar. 2006.

Author Index

Keyword Index