Hacia la generación automática de mecánicas de juego: un editor de reglas para Eryna

Alejandro Ruiz-Moyano, Mariela Nogueira-Collazo, Antonio J. Fernández-Leiva

Departamento de Lenguajes y Ciencias de Computación, Universidad de Málaga
axruizm@gmail.com, {mnogueira, adfez}@lcc.uma.es

Resumen. Este artículo es parte de un trabajo emergente dirigido a la creación automática de juegos de estrategia en tiempo real (RTS), lo cual incluye la generación de contenido, la creación de la inteligencia artificial del juego (es decir, de los jugadores virtuales) y de la propia mecánica del mismo. En este artículo se describe un editor que consta de múltiples parámetros que pueden configurarse para activar o desactivar decenas de reglas que pueden cambiar la mecánica de un juego RTS. El trabajo aquí descrito se realiza en el contexto del juego Eryna, una plataforma de experimentación implementada en la Universidad de Málaga. El editor de reglas es una herramienta que será usada en el futuro para permitir la evolución automática de las reglas de este videojuego y la creación de distintas mecánicas para el mismo. Lo que presentamos aquí debe ser evaluado como un trabajo emergente en el cual estamos trabajando.

Palabras claves: Generación Automática de Contenido, Evolución de reglas, Juegos RTS.

1. Introducción y motivación

El videojuego es un producto cultural que ha mostrado una proyección de crecimiento enorme en los últimos años. Es un sector que lleva varios años produciendo más actividad económica que el resto de sectores culturales y que sigue creciendo exponencialmente debido a su excelente aceptación [DEV, 2015]. Según diferentes consultoras, el mercado de los videojuegos seguirá creciendo sostenidamente a lo largo de los próximos años. Por ejemplo, la consultora PWC estima que en 2019 el mercado global de videojuegos ascenderá a 93.180 millones de dólares [PWC, 2015].

Si se analizan las tendencias modernas del desarrollo de videojuego, una de las líneas de investigación más interesantes consiste en la Generación de Contenido por Procedimientos (PCG, por sus siglas en inglés, Procedural Content Generation), que se puede definir como la automatización de la creación de contenido para los videojuegos. PCG es capaz de agilizar y facilitar el trabajo de los desarrolladores así como de reducir costes en el proceso de desarrollo de un videojuego. El contenido, generado

algorítmicamente, puede ser diverso desde mapas y niveles, estructura de las armas, narrativa, o incluso la propia mecánica [Shaker et al., 2015].
Este artículo es parte de un trabajo preliminar en el cual los autores buscan la creación de juegos completos de forma automática. En los últimos años hemos propuesto modelos coevolutivos para generar automáticamente comportamientos para los oponentes (virtuales) en juegos de estrategia en tiempo real (RTS, por sus siglas en inglés, Real Time Strategy Games) [Nogueira et al., 2014b]; posteriormente propusimos un modelo general y novedoso para coevolucionar, al mismo tiempo, estrategias y contenido [Nogueira et al., 2016]. Ahora queremos cerrar el círculo proponiendo un esquema para la generación automática de mecánicas de juegos en juegos RTS (con el objetivo de generalizarlo luego a otros géneros de juegos). Si avanzamos en este sentido, una correcta modularización de todas nuestras propuestas podría dar lugar a la generación completa de juegos RTS (es decir, de contenido para el mismo, de estrategias de inteligencia artificial para los jugadores virtuales no controlados por el jugador humano, e incluso de la propia mecánica o reglas del juego), con excepción de la música y la sonorización del juego. Es un trabajo complejo pero creemos que es posible puesto que ya se han propuesto trabajos en la literatura científica para automatizar la propia generación de reglas/mecánicas de juegos básicos – tipo puzle [Shaker et al., 2015].
Este artículo debe ser evaluado como un trabajo emergente, como un proyecto preliminar que se ha llevado a cabo para acometer una tarea mucho más complicada. El marco de trabajo es el juego Eryna [Nogueira et al., 2014a] en el cual estamos investigando la generación de distintas mecánicas que darán lugar a otras versiones de un juego original que actúa como semilla. En este artículo se describe un editor de reglas que se ha implementado como una primera aproximación a este objetivo. La idea es mostrar, en el seno del CoSeCiVi, una idea mucho más general.

2. Eryna y el editor de reglas

Eryna, es un videojuego RTS desarrollado en la Universidad de Málaga que sirve también como herramienta de apoyo, y de banco de pruebas, a los investigadores en el campo de la Inteligencia Artificial (IA), y que pone a su disposición módulos relacionados con los videojuegos adaptativos, videojuegos afectivos y aplicaciones de PCG. Nace de la necesidad de investigar las nuevas tendencias en el ámbito de la IA aplicada a los videojuegos. Con este objetivo en mente, Eryna está siendo diseñado/implementado para ofrecer ciertas facilidades a los investigadores de este campo como, por ejemplo, el guardado de logs fáciles de procesar para analizar resultados. Los fundamentos de la implementación de Eryna fueron proporcionados en [Nogueira et al., 2014a]; a continuación describimos una aplicación que se ha creado para editar reglas de este juego de manera sencilla e intuitiva. Esta aplicación es la base para la futura evolución automática de reglas en el propio juego (trabajo que acometeremos en un futuro cercano). A continuación se describe brevemente el juego Eryna y posteriormente nos enfocamos en el editor de reglas.

2.1 Escenario de juego Eryna

Se trata de un videojuego RTS multijugador que simula un entorno galáctico donde el objetivo de los jugadores es conquistar el mayor número de planetas. Los jugadores pueden enviar tropas a los diferentes mundos y las batallas se generan cuando tropas provenientes de distintos jugadores se encuentran en un mismo planeta, pues tienen que luchar entre sí y solo uno logrará la conquista de ese mundo.
Inicialmente este juego tenía su lógica integrada dentro de la propia game engine, por lo que se le daba al juego un comportamiento estático que limitaba las simulaciones a mecánicas estándares. Esta situación suponía un problema a la hora de implementar el editor de reglas, ya que éstas no admitían edición dinámica en absoluto. Por ello, se llevó a cabo un rediseño de la arquitectura del juego (separando la lógica del resto del juego) para posibilitar la parametrización de sus reglas pudiendo, así, modificarlas de forma sencilla.
El editor de reglas es un subsistema independiente del proyecto Eryna pero que se relaciona estrechamente con él, permitiendo generar y editar parte de las reglas y recursos del juego que definen el entorno de la partida. Se han predefinido un buen número de parámetros asociados a la activación/desactivación de las posibles reglas del juego, y se permite que el usuario defina nuevas reglas y nuevos parámetros. El usuario (investigador) que usa el editor de reglas generará un fichero con los parámetros que haya introducido en él. La mecánica del juego, definida por las reglas que definen la lógica, será el resultado de la combinación de los valores de esos parámetros. Digamos que los parámetros no sólo pueden activar o desactivar reglas predefinidas sino que también pueden influenciar los estados iniciales e intermedios de cada partida (por ejemplo, diferentes porcentajes de crecimiento para los planetas pueden hacer que la partida se alargue más o menos).

2.2 Formato de las reglas

Los lenguajes de descripción de juegos o GDL (Game Description Language, por sus siglas en inglés) son altamente útiles en este aspecto ya que nos permiten detallar las reglas de los juegos de forma unívoca y eficaz [Mahlmann et al., 2011]. Como ejemplos de estos lenguajes se pueden destacar varios con diferentes enfoques. Por ejemplo, el Stanford GDL (SGDL) es un lenguaje relativamente independiente de los géneros de juego pero está limitado a juegos con espacio de estado discreto y sus descripciones tienden a ser bastante largas debido a que se basa en la lógica de primer orden [Love, 2006]. El Ludi GDL, propuesto en contraposición al Stanford GDL, se limita a juegos de mesa de 2 jugadores con restricciones de elementos y tableros para ser mucho más conciso en las descripciones [Browne, 2008]. Por último cabe mencionar el Ludocore, el cual expresa juegos arcade en 2D usando programación lógica [Smith, 2010].
El GDL que nos parece más interesante para nuestro editor de reglas es el propuesto por Mahlmann et al. Este lenguaje está completamente centrado en los juegos RTS y es relativamente fácil de leer por un humano. La desventaja principal es que para el

tipo de reglas que se necesitan especificar en este proyecto es un tanto engorroso. No obstante, Mahlmann et al. especifican para su SGDL tres capas en las que se pueden dividir los juegos de tipo RTS:

- Capa de mecánicas: determina las reglas fundamentales del juego, como por ejemplo las acciones de ataque o el tipo de escenario que usa el juego.
- Capa de ontología: especifica los elementos clave que pueden existir en el juego y sus propiedades.
- Capa de instancia: detalla la organización específica de un mapa o de un nivel del juego.

Siguiendo esta definición por capas, nos hemos centrado en la capa de mecánicas, ya que el objetivo es modificar las mecánicas de juego mediante el editor y, luego, facilitar la creación de variaciones de estas mecánicas mediante un algoritmo de inteligencia artificial avanzada. Como primera aproximación a estos objetivos hemos empleado un lenguaje de descripción de reglas más simple que SGDL, y se ha optado por seguir los pasos del sistema ANGELINA [Cook, 2011] para este proyecto. ANGELINA es un sistema que sirve para crear juegos de género arcade desde cero, permitiendo la creación de reglas, mapas de terreno y estableciendo la posición inicial de los jugadores. Para este propósito, usaron el formato XML para representar las reglas y las características de los juegos. De esta forma, los archivos que produce (y que puede cargar) nuestro editor de reglas para Eryna tienen el formato XML y/o JSON, ya que ambos son muy estructurados, extensibles y fáciles de leer por un ser humano a simple vista.

2.3 Reglas básicas y los parámetros que las configuran

Ahora que se ha determinado el formato para las reglas del juego, resulta imprescindible definir unas reglas básicas y unos parámetros asociados a ellas que puedan influir en las mismas, bien activándolas o desactivándolas (lo cual quiere decir que formarán parte o no de la lógica del juego) o dotándolas de ciertas características variables (por ejemplo, una regla de realizar una acción de invadir un planeta puede depender del número de naves enemigas que haya en ese planeta, y ese número es un parámetro variable de la regla; si consideramos un valor pequeño las estrategias de los jugadores se volverán más agresivas que si consideramos un número elevado). Con la inclusión de rangos para los parámetros asociados a las reglas se pretende acotar el alcance de los cambios en las reglas para que éstos no conduzcan a situaciones indeseadas y, además, facilitar al usuario el uso de la herramienta de edición como forma de abstracción de las reglas en sí mismas.

Las reglas predefinidas inicialmente se dividen en varias categorías atendiendo a la función que cumplen dentro del juego. Las categorías disponibles y sus correspondientes reglas son las que se especifican a continuación:

Misiones

La primera de las categorías de reglas del juego son las llamadas misiones: acciones de envío de tropas de un planeta a otro con diferentes funciones y restricciones. El juego dispone de tres tipos de misiones base:

- Conquista: acción de enviar tropas a un planeta neutral.
- Invasión: acción de enviar tropas a un planeta cuyo dueño es otro jugador con la finalidad de arrebatárselo.
- Expansión: acción de enviar tropas a un planeta aliado para potenciarlo.

Asociados a cada una de estas reglas potenciales para el juego, existen parámetros booleanos que indicarán si esa regla forma parte de las mecánicas del juego o no. Adicionalmente, para cada regla asociada con las misiones, existe un conjunto adicional de parámetros que pueden variar igualmente la mecánica del juego: Constructores, Tanques, Barcos, Aviones, Soldados y Trabajadores. En todos los casos, estos parámetros contendrán la cantidad máxima de la unidad de ese tipo que se puede enviar en la misión. También existe un flag/parámetro que elimina la restricción de que existan valores limitados en esos parámetros (si se activa este flag el espacio de búsqueda incrementa enormemente). La Figura 1 (arriba) muestra una pantalla de edición de nuestro editor desde la cual se pueden configurar las misiones del tipo invasión; la Figura 1 (abajo) muestra dos pantallas asociadas con la edición de nuevas reglas (es decir, de reglas que son predefinidas).

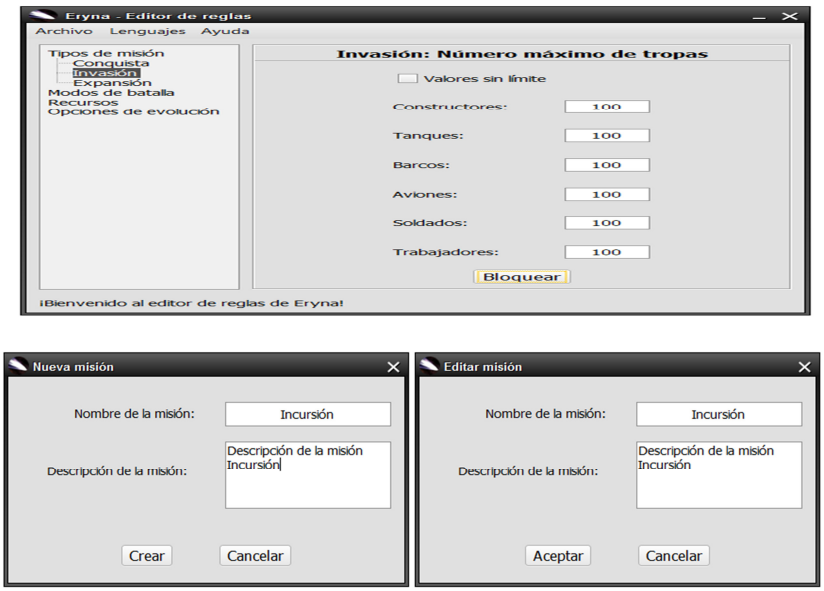

Fig 1. El editor de reglas. (arriba) Edición de misiones tipo conquista. (abajo) Edición de nuevas reglas de tipo misión.

Batallas

Las batallas en el juego se producen cuando se realiza el envío de tropas a un planeta no aliado y la llegada de las mismas coincide con la llegada de tropas de otro jugador (o jugadores) al mismo planeta. De este modo, una de las reglas modificables del

juego es el modo de batalla que éste ejecutará cuando se produzca el caso anterior. El usuario puede escoger entre 4 opciones predeterminadas posibles y no puede crear una por su propia cuenta (a diferencia de las misiones). Estas opciones predeterminadas se manipulan mediante los parámetros booleanos especificados a continuación:

- *El más fuerte:* sólo los dos jugadores más fuertes luchan entre sí y el resto son eliminados del combate.
- *Torneo de eliminación directa:* cada jugador lucha contra otro jugador escogido aleatoriamente. El perdedor de cada encuentro es eliminado del combate y el vencedor lucha contra otro vencedor. Este ciclo se repite hasta que solamente quede un vencedor.
- *Torneo por grupos:* los jugadores son organizados por grupos aleatoriamente. Cada jugador lucha contra los demás jugadores del mismo grupo, ganando puntos en función de los resultados de cada combate. El jugador con más puntos es el vencedor del grupo y, luego, los vencedores se enfrentan entre ellos mediante torneos con el sistema de eliminación directa.
- *Aleatorio*: se seleccionan aleatoriamente dos jugadores para luchar entre sí y el resto son eliminados del combate.

Recursos

A esta categoría pertenecen las reglas relacionadas con los recursos disponibles del juego, los cuales pueden ser bióticos o industriales. Con respecto a los recursos bióticos, el diseñador puede configurarlos mediante los parámetros siguientes:

- *Fuerza de trabajador:* determina el nivel de fuerza de combate por defecto que tienen las unidades de tipo trabajador.
- *Fuerza de soldado:* determina el nivel de fuerza de combate por defecto que tienen las unidades de tipo soldado.
- *Hambre***:** determina el nivel de hambre por defecto de todas las unidades bióticas (trabajador y soldado).
- *Libido:* determina el nivel de libido por defecto de todas las unidades bióticas (trabajador y soldado). Tiene influencia en el factor de crecimiento de cada planeta.

Los valores de hambre y de libido afectan a la eficacia de las tropas en su labor, y los valores de fuerza se usan para los combates (en el caso de los recursos industriales el uso es el mismo).

Los recursos industriales, por su parte, pueden ser configurados manipulando los siguientes parámetros: *Fuerza de máquina constructora, Fuerza de nave aérea, Fuerza de nave acuática,* y *Fuerza de nave terrestre*. Cada uno de estos parámetros determina el nivel de fuerza de combate por defecto de las unidades consideradas.

Evolución

Por último, en esta categoría se hallan las reglas relacionadas con la evolución de las diferentes unidades del juego (en el caso de las unidades industriales hablamos de desgaste en lugar de evolución). Además, también se pueden modificar las reglas de

evolución de los recursos naturales que se encuentran en los planetas. Las reglas modificables, tanto para seres vivos como para recursos industriales como para recursos naturales, están influenciadas por los siguientes parámetros:

- *Evolución de seres vivos:* permite activar y desactivar la evolución de los diferentes seres vivos en el juego.
- *Factor evolutivo de seres vivos:* si la regla de evolución de los seres vivos está activa, indica el factor de evolución de los mismos.
- *Factor evolutivo de bestias:* si la regla de evolución de los seres vivos está activa, indica el factor de evolución de las bestias neutrales de los planetas.
- *Factor de recuperación:* si la regla de evolución de los seres vivos está activa, indica el factor con el que las unidades vivas reducirán sus niveles de hambre y libido y aumentarán su nivel de fuerza con el paso del tiempo.
- *Cansancio de vuelo:* permite activar y desactivar el cansancio de vuelo para las unidades vivas, lo que afectará a su rendimiento en combate.
- *Factor de reducción de fuerza:* si la regla de cansancio de vuelo está activa, indica el factor de reducción que se le aplicará al valor de fuerza de las unidades vivas durante los vuelos.
- *Desgaste de recursos industriales:* permite activar y desactivar el desgaste de los recursos industriales.
- *Factor de desgaste de recursos industriales***:** si la regla de desgaste de recursos industriales está activa, indica el factor de desgaste que sufrirán dichos recursos con el paso del tiempo.
- *Evolución de recursos naturales:* permite activar y desactivar la evolución de los recursos naturales pertenecientes a los planetas.
- *Factor evolutivo de recursos naturales:* si la regla de evolución de recursos naturales está activa, indica el factor con el que evolucionarán los recursos naturales.

La variación en los valores de los parámetros asociados a cada tipo de reglas nos conduce a diferentes mecánicas del juego. En total se disponen de 32 parámetros que activan/desactivan reglas del juego o tienen influencia en éstas.

3. Experimentos preliminares

A partir de las reglas anteriores y los parámetros que las activan o las influencian, la idea es permitir la evolución de las reglas del juego usando técnicas de PCG. Esa generación puede estar encaminada a potenciar mecánicas de juegos específicas. Una ventaja del juego utilizado y de su fácil parametrización es que se pueden crear entornos lógicos diversos según la configuración establecida.

Como punto de partida a una fase futura de explotación y optimización de la generación automática de reglas, se realizaron varios experimentos para probar la viabilidad de generar dichas reglas usando algoritmos genéticos convencionales. Las funciones de *fitness* definidas fueron dos:

(a) La primera potenciaba la búsqueda de reglas que generasen partidas balanceadas. Que una partida esté balanceada quiere decir, en el caso de *Eryna*, que la diferencia entre la cantidad de planetas que poseen los jugadores nunca supera un umbral Θ determinado que la haga significativa.

(b) La otra función de *fitness* encaminaba la búsqueda hacia soluciones que aportaran dinamismo a la partida. En el contexto abordado, se entiende por dinamismo la cantidad de veces que el jugador que lleva la ventaja (el que más planetas tiene) cambia de objetivo, es decir, cambia de planeta destino para enviar sus misiones.

Para la elaboración de estos experimentos se diseñaron tres algoritmos genéticos (AG) que generan reglas de juego y las evalúan atendiendo a tres criterios diferentes: maximizar la cantidad de turnos en que la partida se considera que está balanceada, maximizar la cantidad de turnos en que la partida se considera que es dinámica y maximizar los dos casos anteriores, dándoles un peso de importancia del 50% a cada uno. Cada cromosoma consiste en un número de genes que representan los parámetros asociadas a cada una de las mencionadas anteriormente. Cada AG fue diseñado como un algoritmo generacional con la siguiente parametrización: número de generaciones = 30, *popzize* = 15, selección por torneo binario, probabilidad de cruce = 99%, y probabilidad de mutación = 1/(número de genes). Con respecto a las funciones de fitness impusimos los siguientes valores: umbral Θ = 20% -para el *fitness* (a)-, y maximizar la cantidad de veces que cambia el jugador que va ganando - para el *fitness* (b).

Para obtener unos datos sólidos, se emplearon los mismos bots y el mismo mapa en todos los experimentos realizados. Los 2 bots que se enfrentaban en cada caso seguían un patrón de ataque completamente aleatorio y el mapa estaba compuesto por 9 planetas (7 neutrales y 1 para cada bot al inicio).

En la Figura 2 se puede observar las gráficas obtenidas con respecto a los dos primeros experimentos mencionados, en los cuales valores de *fitness* se corresponden con la cantidad de turnos que cumplen el objetivo propuesto en una partida multiplicados por 100. Esto nos proporciona un margen amplio de ajuste de los valores de *fitness* en caso necesario. En ambas gráficas se puede observar una tendencia al alza de los valores de *fitness*, por lo que se comprueba que el algoritmo genético resulta eficaz a la hora de evolucionar reglas que persigan los objetivos marcados. En el caso del primer experimento - Figura 2 (izquierda)- se alcanzó un máximo de 6 turnos de balance en las partidas, lo cual quiere decir que en 6 turnos de todos los que se disputaron en la partida la cantidad de planetas que poseían los jugadores estaba balanceada con respecto al umbral fijado del 20%. En el caso del segundo experimento - Figura 2 (derecha)- el máximo número de turnos en los que las partidas fueron consideradas dinámicas creció desde los 3 turnos de las primeras generaciones, hasta alcanzar los 9 turnos de las 10 últimas generaciones.

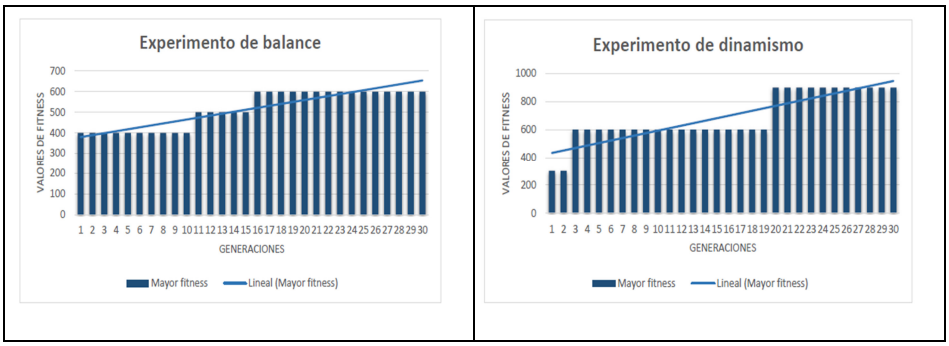

Fig 2. (Izquierda) Mayor *fitness* de partida balanceada (Derecha) Mayor *fitness* de partida dinámica.

En los tres experimentos se observó, sin embargo, que el número de mejoras de los valores de *fitness* no es significativo. Creemos pues que es necesario afinar aún los parámetros del algoritmo para obtener unos resultados más satisfactorios.

En cualquier caso, estos experimentos preliminares nos sirven como demostración de que se pueden generar mecánicas de juegos muy diversas al combinar las opciones de parametrización.

4. Discusión y trabajo futuro

Este artículo describe un paso más en nuestro objetivo de crear juegos completos de forma automática. En concreto hemos descrito un editor de reglas que hemos creado para la plataforma/videojuego de investigación *Eryna*. Este editor es la base fundamental para permitir la edición de las reglas que conformarán la lógica de un juego RTS y sirve de enlace entre la especificación de ésta y su ejecución dentro del juego.

Se han mostrado los resultados de unos experimentos preliminares que nos indican que es posible realizar la evolución de las mecánicas de juego con fines específicos impuestos por el diseñador y a partir de la optimización de un conjunto de parámetros que influencian la lógica del juego (a través de las reglas del mismo).

Este artículo es un trabajo preliminar y queda todavía un largo camino por recorrer. Por ejemplo, nosotros hemos partido de una configuración de juego específica, creada por el diseñador del juego y luego hemos creado variaciones de ésta a través de un mecanismo de optimización evolutivo. Está claro que esta configuración inicial va a influir en las posibles variaciones del juego a obtener. Por un lado es algo positivo pues el diseñador toma el control del punto de partida de la evolución y puede ser vista como una sugerencia a partir de la cual quiere o desea obtener inspiración. Por otra parte, esta semilla inicial limita claramente la capacidad creativa del proceso de desarrollo automático de contenido. El cómo crear esa configuración inicial que sirve de semilla para la evolución del juego y el análisis de su influencia sobre las versiones que se generan es una línea que debemos abordar en un futuro próximo.

5. Agradecimientos

Este trabajo está financiado parcialmente por la Junta de Andalucía (proyecto P10-TIC-6083 DNEMESIS[1], el Ministerio Español de Economía y Competitividad (proyecto TIN2014-56494-C4-1-P, UMA-EPHEMECH[2], y la Universidad de Málaga. Campus de Excelencia Internacional Andalucía Tech.

6. Referencias

[Browne, 2008] Browne, C. (2008). Automatic generation and evaluation of recombination games. Ph.D. thesis, Queensland University of Technology.

[Cook, 2011] Cook, M. & Colton, S. (2011). Multi-faceted evolution of simple arcade games. IEEE Conference on Computational Intelligence and Games, 289-296.

[DEV, 2015] Asociación Española de Empresas Desarrolladoras de Videojuegos y Software de Entretenimiento (2015). Libro Blanco del Desarrollo Español de Videojuegos.

[Love, 2006] Nathaniel C. Love, Timothy L. Hinrichs, and Michael R. Genesereth. General Game Playing: Game Description Language specification. Technical report, LG-2006-01, Stanford Logic Group, 2006..

[Mahlmann et al., 2011] Mahlmann, T., Togelius, J. & Yannakakis, G. N. (2011). Towards Procedural Strategy Game Generation: Evolving Complementary Unit Types. EvoApplications (1), 93-102.

[Nogueira et al., 2014a] Nogueira, M., Cotta, C., & Fernández, A. J. (2014). Eryna: una herramienta de apoyo a la revolución de los videojuegos. CoSECivi, (págs. 173-184). Barcelona.

[Nogueira et al., 2014b] Mariela Nogueira Collazo, Carlos Cotta, Antonio José Fernández Leiva: Virtual player design using self-learning via competitive coevolutionary algorithms. Natural Computing 13(2): 131-144 (2014)

[Nogueira et al., 2016] Mariela Nogueira Collazo, Carlos Cotta, Antonio José Fernández Leiva. Competitive Algorithms for Co-evolving both Game Content and AI. A Case Study: PlanetWars. IEEE Transactions on Computational Intelligence and AI in Games. Aceptado para su publicación (2016)

[PWC, 2015] PWC. (2015). Obtenido de: http://www.pwc.com/gx/en/industries/entertainment-media/outlook/segment-insights/video-games.html

[Shaker et al., 2015] Shaker, N., Togelius, J., Nelson, M.J.: Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer (2015)

[Smith, 2010] Smith, A. M. & Mateas, M. (2010). Variations Forever: Flexibly generating rulesets from a sculptable design space of mini-games. IEEE Conference on Computational Intelligence and Games, 273-280.

[1] http://dnemesis.lcc.uma.es/wordpress/

[2] https://ephemech.wordpress.com/