Optimización evolutiva de bots para el juego Planet Wars

A. Fernández-Ares, A.M. Mora, P. García-Sánchez, J.J. Merelo, C. Fernandes

Resumen— **Este artículo describe un algoritmo evolutivo (AE) para la optimización del motor de decisión de un programa que, en este contexto, se denominará *bot*, diseñado para jugar al videojuego Planet Wars, en el que dos jugadores se enfrentan en combates por turnos para dominar una serie de planetas contando con sendas flotas de naves. Dicho juego fue elegido como entorno para acoger el Google Artificial Intelligence Challenge 2010, una competición internacional en la que se proponía a los participantes el desarrollo de un bot autónomo que fuese capaz de ganar el mayor número de enfrentamientos en dicho juego contra otros rivales automáticos. Por tanto, el bot debía saber gestionar sus recursos y tener capacidad de adaptación ante los diferentes "campos de batalla estelares" y posibles rivales. Tras un estudio empírico se definió un conjunto de reglas que modelaban el comportamiento deseado para el bot. Después se aplicó un algoritmo genético (AG) para refinar una serie de parámetros de los que dependían las reglas y, por tanto, el comportamiento del bot. Este trabajo describe la aplicación de dicho algoritmo y presenta un análisis de los resultados obtenidos en enfrentamientos entre dicho bot 'genético' y otros dos (el estándar de Google y uno creado siguiendo reglas fijas). La naturaleza variable del fitness y su influencia en los resultados es también estudiada en los experimentos.**

Palabras clave— **Juegos de Estrategia en Tiempo Real, ETR, RTS, Algoritmos Evolutivos, Algoritmos Genéticos, Bots, Optimización del Comportamiento, Agentes Autónomos, Inteligencia Artificial**

I. Introducción

Los *Bots* [25] son agentes autónomos que interaccionan con un usuario humano dentro de un entorno de computación, como por ejemplo los videojuegos. En ellos, los bots intentan actuar siguiendo pautas de cooperación o sobre todo de competición respecto al jugador humano, intentando incrementar el desafío que ofrece el juego y, por tanto, su diversión. En este sentido, su *comportamiento* es uno de los parámetros fundamentales en el diseño de los juegos [11].

Los bots se usan principalmente dentro de los juegos de disparo en primera persona (o First Person Shooters, FPS, en inglés) [4], [18], en los que se diseñan como oponentes para los jugadores humanos siguiendo pautas de Inteligencia Artificial (IA). En este artículo, en cambio, trabajaremos dentro del entorno de un juego de Estrategia en Tiempo Real, ETR (Real-Time Strategy Game o RTS en inglés), que se encuadra dentro de los juegos de estrategia, en el que los contendientes controlan un conjunto de unidades y estructuras distribuidas en un mapa de juego las cuales se deben enfrentar en combates para doblegar al rival.

Un juego de ETR típico ofrece al jugador la posibilidad de crear unidades y estructuras adicionales durante la batalla, aunque con ciertas restricciones, como contar con una serie de recursos limitados que se invierten en la creación de dichos elementos, si bien, es posible hacer acopio de más recursos en el campo de combate. Otra de sus propiedades es, precisamente, su naturaleza de juego en tiempo real, lo que significa que un jugador no debe esperar a las acciones y resultados de los demás para realizar las propias (como sí se haría en un juego por turnos). Algunos ejemplos famosos de este tipo de juegos son: Command and Conquer™, Starcraft™, Warcraft™ o Age of Empires™ .

Google eligió un juego de este tipo para el desarrollo de su AI Challenge 2010 [6], *Planet Wars* (Galcon [26] originalmente). En esta competición, el tiempo real se dividió en *turnos* de un segundo, a fin de que los jugadores pudiesen jugar de forma secuencial, sin embargo las *acciones* se suceden al mismo tiempo en la simulación, convirtiéndose en una peculiaridad de la implementación, no en una característica del juego (que lo convertiría en un juego de turnos).

En este artículo se propone un enfoque evolutivo para la creación del motor de decisión de un bot que juegue a *Planet Wars* (o Galcon [26]). Este enfoque fue implementado en dos pasos: en primer lugar, se diseñaron mediante pruebas exhaustivas un conjunto de reglas que modelasen el comportamiento del bot en función de una serie de variables sobre dichas reglas; el segundo paso consistió en aplicar un Algoritmo Genético (AG) [3] para la evolución de las variables fuera del entorno de la competición (es decir, como un proceso previo al campeonato).

La evaluación de la calidad (fitness) de cada conjunto de reglas o bot de la población ha sido calculado realizando simulaciones de juegos enteros entre dicho bot y un oponente predefinido. Partiendo del fitness de cada individuo y siguiendo el paradigma clásico de los AGs, se han aplicado a la población un determinado número de generaciones evolutivas. Vamos a demostrar como la población del conjunto de reglas o bots evolucionan hacia bots mejores, hasta finalmente devolver un bot más eficaz y eficiente.

Además, como aporte adicional se realizarán experimentos para analizar el problema del llamado 'fitness ruidoso' que aparece en este tipo de problemas, es decir, un mismo individuo podría tener asignado un fitness muy bueno en una iteración y muy malo

Departamento de Arquitectura y Tecnología de Computadores. Universidad de Granada
{antares,amorag,pgarcia,jmerelo,cfernandes}@geneura.ugr.es

en otra, dado que dicho fitness depende de la evaluación de su comportamiento en el enfrentamiento contra otro rival (y posiblemente en otra situación de juego), lo que significaría un resultado diferente para la partida.

El resto del artículo se estructura como se comenta a continuación. En la sección siguiente se describe el entorno del juego Planet Wars, así como sus características. La Sección III presenta los principales estudios relacionados con el tema que centra la presente investigación. Los enfoques y algoritmos propuestos para la generación del bot se describen en la Sección IV. A continuación los experimentos, resultados y su análisis se muestran en la Sección V. Finalmente, la Sección VI presenta las conclusiones y futuras líneas de trabajo a seguir.

II. Planet Wars

El *Google AI Challenge (GAIC)* [6] es una competición de IA en la que los participantes diseñan bots para competir los unos contra los otros. El juego elegido para la competición de 2010, *Planet Wars*, es el entorno de estudio elegido en este artículo. Para este juego, hemos propuesto diseñar el conjunto de reglas que definen comportamiento del bot y mediante el uso de AGs optimizar su eficacia y eficiencia. *Planet Wars*, es una versión simplificada del juego Galcon [26], donde los enfrentamientos involucran únicamente a dos jugadores.

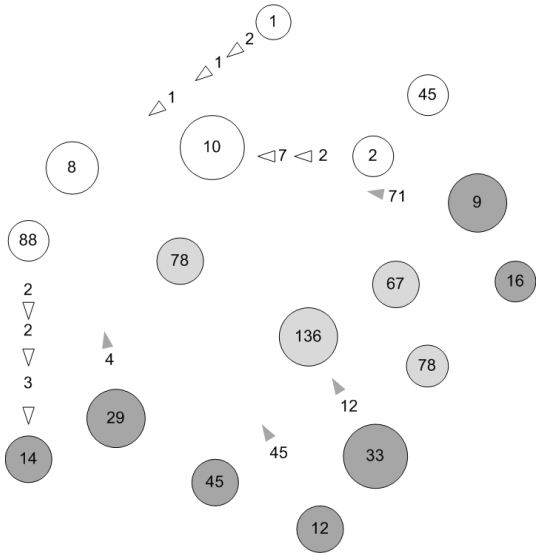

Fig. 1.
Captura de la simulación de un estado temprano del juego Planet Wars. Los planetas blancos pertenecen al jugador, los planetas grises oscuros pertenecen al oponente y los planetas grises claros no pertenecen aún a ningún jugador. Los triángulos representas flotas de naves. Los números (tanto en planetas como en flotas) representan el número de naves que lo componen.

La contienda entre los jugadores tiene lugar en un mapa o escenario de juego que alberga varios planetas, cada uno con un número asignado que representa la cantidad de naves que están alojadas en él (ver Figura 1). En cada instante de tiempo, cada planeta alberga una cantidad específica de naves, que pueden pertenecer al jugador, al oponente o ser neutrales. La propiedad está representada por un color asignado a cada jugador. Además, cada planeta tiene una tasa de crecimiento que indica cuantas naves se generan durante cada asalto y son agregadas a la flota que el jugador posee en el planeta. Los planetas neutrales, no agregan nuevas a la flota que alojan hasta que el planeta es conquistado por algún jugador.

El objetivo del juego es apoderarse de todos los planetas del rival. Aunque Planet Wars es un RTS, la implementación en el campeonato lo ha transformado en un juego basado en *pseudo-turnos*, en donde los jugadores disponen de un tiempo (número de turnos) determinado para conseguir la victoria. Cada pseudo-turno dura un segundo, de modo que ese es el tiempo de que dispone un bot para realizar sus acciones. Otra particularidad del problema es que no está permitido almacenar ningún tipo de información sobre las acciones propias anteriores, las del oponente, ni sobre el estado de juego (el mapa). En otras palabras, en cada pseudo-turno de un segundo el bot debe hacer frente a un mapa desconocido tal y como si fuese un juego nuevo. Esta restricción, hace que el desarrollo del bot sea un reto realmente interesante.

De hecho, cada bot se implementa como una función que, partiendo de una lista de planetas y sus flotas (el estado actual del mapa), devuelve una lista de acciones a realizar. En cada pseudo-turno, el bot debe elegir el movimiento que realizarán una o más flotas desde un planeta propio a cualquier otro planeta. Esta acción es la única que puede llevar a cabo el bot. Las flotas, pueden necesitar más de un pseudo-turno en llegar a su planeta de destino (el tiempo es directamente proporcional a la distancia entre el planeta de origen y el destino). Cuando la flota llega un planeta enemigo o neutral, tiene lugar un enfrentamiento, en el cual cada nave es sacrificada para destruir una nave enemiga, en otras palabras, resulta victoriosa aquella flota que albergue más naves. En caso de que el planeta de destino sea del propio jugador, ambas flotas se unen, sumando sus naves. En cada pseudo-turno, el número de naves alojadas en los planetas de los jugadores (no los neutros), se incrementa de acuerdo a la tasa de crecimiento de cada planeta.

III. Estado del Arte

Los videojuegos se han convertido en uno de los mayores y más importantes sectores de la industria del ocio. La mayoría de las empresas involucradas en el desarrollo de videojuegos, han centrado sus esfuerzos e inversiones en el aumento de la calidad gráfica, en lugar de intentar innovar ofreciendo un mayor desafío a las habilidades del jugador. Dada esta tendencia, los jugadores cada vez se ven menos sorprendidos y atraídos por el matiz gráfico, dirigiendo

su atención hacia otros aspectos, como el de ofrecer oponentes (o colaboradores) controlados por la máquina que realicen comportamientos inteligentes similares (o incluso superiores) a los realizables por un oponente (o colaborador) humano[12].

Durante los últimos años, la investigación en el campo de la IA en videojuegos, se ha centrado en juegos FPS como Doom^{TM} , Quake^{TM} [11] o Unreal Tournament ^{TM} [22], [4], [18]; así como en juegos relativamente simples, como Super Mario [24], Pac-Man [15] o juegos de conducción [19].

Aún así, las investigaciones dedicadas a la IA de juegos RTS están en un estado emergente. Las anteriormente citadas características de este tipo de juegos hacen que sean más atractivos para jugadores más experimentados, así como más interesantes para un investigador, dado que son muchos los problemas involucrados en el diseño de IAs para los juegos RTS, como por ejemplo: problemas de planificación con información incompleta, aprendizaje, modelos competitivos y colaborativos, o búsquedas espaciales y temporales.

Sin embargo, la realidad actual refleja que la mayoría de los bots en juegos de estrategia en tiempo real están controlados por guiones (scripts) fijos que han sido previamente programados (por ejemplo a raíz de una máquina de estados finitos o un árbol de decisiones), lo que ocasiona que los jugadores humanos terminen conociendo las rutinas de comportamiento de sus oponentes no humanos, prediciendo los actos de la máquina y actuado en consecuencia, lo que hace que el juego pierda rápidamente su atractivo. Con el fin de mejorar esto, algunos autores como Falke y otros [5] proponen que los bots se ajusten y cambien dinámicamente sus propias estrategias en respuesta a las acciones de sus oponentes.

Además en muchos RTS las técnicas tradicionales en IA fallan al intentar asemejarse a las conductas de un ser humano, dado el enorme espacio de búsqueda en el que se ven involucrados estos juegos. En este sentido, Ontano y otros [20] proponen extraer en forma de casos individuales el comportamiento realizado por expertos. De igual forma, recientemente se están utilizando técnicas de soft-computing y algoritmica, como algoritmos co-evolutivos [17], [2], [10] o multiagentes [8] para manejar estos problemas en los RTS, dadas las múltiples ventajas existentes al diseñar IAs con aprendizaje adaptativo que puedan existir a varios niveles de jerarquía y/o que co-evolucionan con el tiempo. En estos casos, las estrategias de co-evolución pueden ir incluso más allá de los enfrentamientos, como la colaboración a distintos niveles [13] o la evolución de tácticas en equipo [1].

Los AEs han sido también utilizados anteriormente en este campo [21], [9], pero su elevado coste computacional hace que su uso on-line en los juegos no sea muy frecuente. De hecho, las propuestas más exitosas de AEs en los juegos corresponden con su utilización off-line [23], es decir la aplicación de un algoritmo evolutivo que opera (por ejemplo para optimizar el conjunto de reglas que determinan las acciones del bot) mientras las partidas no son jugadas, con el fin de utilizar los resultados de las mejoras obtenidas por el AE durante los juegos. A través del aprendizaje evolutivo off-line, se puede mejorar la calidad de la IA de los bots de los juegos comerciales, y ha demostrado ser más eficaz que un entorno basado en guiones fijos.

Siguiendo esta vía y teniendo presentes las limitaciones impuestas por el campeonato (el bot no tiene conocimiento previo ni memoria y cada pseudo-turno está limitado a un segundo), en este trabajo se aplica un Algoritmo Genético off-line para optimizar las variables que parametrizan el conjunto de reglas que modelan el comportamiento del bot en el juego Planet Wars, con el objetivo de obtener un conjunto determinado de reglas o bot que pueda ser utilizado durante las partidas en tiempo real.

IV. Evolución de Bots en Planet Wars

En la Sección II se ha comentado que la mayor dificultad del entorno es la limitación temporal para decidir las acciones a emprender (un segundo) y la prohibición de utilización de memoria. Estas restricciones limitan las posibilidades de diseño e implementación del algoritmo que rija el comportamiento del bot, pues muchas metaheurísticas están basadas en el uso de una memoria para almacenar y puntuar las acciones anteriores, con el fin de mejorar el comportamiento futuro. Además, muchas de estas metaheurísticas son también muy costosas computacionalmente hablando. Por ejemplo, ejecutar un AE o hacer uso del método de Monte Carlo [14] en un segundo es prácticamente imposible. Además solo se puede evaluar globalmente una estrategia, es decir, hasta que no ha finalizado la partida, no podemos deliberar si la estrategia empleada ha sido satisfactoria o no. En definitiva, no es posible evaluar y optimizar acciones individuales dentro de una partida, sino todo la partida en si.

Estas razones, han propiciado el establecimiento de un conjunto de reglas que modelase el comportamiento de la IA del bot durante las partidas. Estas reglas han sido constituidas a través de una experimentación exhaustiva, y son muy dependientes de una serie de parámetros que definen y determinan el comportamiento del bot.

En cualquier caso, sólo existe un tipo de acción permitida: el movimiento de flotas de un planeta a otro y aunque la naturaleza de la acción es relativamente sencilla, las decisiones que implica son bastante complejas: ¿de qué planeta parten las naves? ¿irán a un planeta enemigo, neutral o propio? ¿cuántas naves enviaremos en la flota? ¿a qué planeta las mandaremos? ¿cuántos movimientos de flotas haremos? En definitiva, una única acción pero con innumerables posibilidades.

Durante el campeonato Google proporcionó un

bot ejemplo, para tomar como referencia al que denominamos GoogleBot (presentado en la Sección IV-A). Después se presentará nuestra primera propuesta de bot denominado AresBot (en Subsección IV-B) y finalmente se expondrá el algoritmo y los principales mecanismos que regulan el comportamiento del bot objeto de este estudio, llamado GeneBot (en SubsecciónIV-C).

A. GoogleBot

Se trata de un bot relativamente simple pero muy versátil, diseñado para funcionar bien independientemente del mapa o el comportamiento del rival. Su funcionamiento es el siguiente: para un estado específico de la partida, en primer lugar comprueba si tiene alguna flota en movimiento, de ser así, el bot no haría nada más (esperaría a la llegada de la flota a su objetivo). Si no dispone de flotas en movimiento, establece cual es su planeta más poderoso (aquel que alberga la mayor flota). Dicho planeta será utilizado como origen para una única acción de movimiento. El planeta objetivo será establecido eligiendo aquel planeta (enemigo o neutral) con la mayor tasa, siendo ésta directamente proporcional a la tasa de crecimiento e inversamente proporcional al tamaño de la flota albergada.

A pesar de su simplicidad, GoogleBot resulta un buen oponente, aunque claramente mejorable.

B. AresBot

El primer paso en el diseño de nuestro propio bot fue la implementación manual de un conjunto de reglas que resolviesen los enfrentamientos de forma satisfactoria. Este bot se denominó AresBot y tiene el siguiente funcionamiento: para un estado específico de la partida, el bot localiza su *planeta base* siendo aquel con un mayor número de naves alojadas. El resto de planetas propios se denominan *colonias*. A continuación se determina el *planeta objetivo*, elegido entre los planetas que no pertenecen al jugador (y que el jugador no está intentando conquistar o expandir), y que le aportará un mayor beneficio y/o le implica un menor riesgo. Si el planeta es neutral, se realiza una acción denominada *expansión* y si el planeta es enemigo, la acción se denomina *conquista*. Para toda *colonia* más cercana al *planeta objetivo* que al *planeta base* se produce un envío de flotas al *planeta objetivo*. El bot estima en función de la tasa de crecimiento del *planeta objetivo* y la distancia a la que se encuentra del *planeta base*, el número de tropas que necesitará enviar para garantizar la victoria. Si el número de tropas enviadas desde las *colonias* no es superior a este número (y por tanto, no se tienen garantías de poder conquistar o expandir el *planeta objetivo*) se realiza un envío desde el *planeta base* hasta el *planeta objetivo* con un flota que garantice la victoria. Para el resto de colonias, existe una posibilidad de que se realice un *diezmo*, un 'impuesto' que supone el envío de tropas desde la colonia al planeta marcado como *planeta base*. De esta forma, se obtiene por una parte un *planeta base* que es difícil de capturar por el enemigo dada la gran flota que alberga en él, así como un buen punto de partida desde el que capturar los planetas enemigos.

Se puede estudiar el modelo de conducta de AresBot con mayor detenimiento si se observa el diagrama presente en la Figura 2.

Existe un conjunto de parámetros (consistentes en pesos, probabilidades, cantidades,...) incluidos en las reglas que modelan el comportamiento del bot (como se ve en el diagrama de la Figura 2). Estos valores fueron originalmente establecidos a mano y posteriormente convertidos en parámetros, una vez se comprobó que determinaban el comportamiento del bot. Sus valores y significados se recogen a continuación:

- $tithe_{perc}$: porcentaje de naves que son enviados en un diezmo en función del número de naves albergadas en el planeta.
- $tithe_{prob}$: probabilidad de que una colonia determinada envíe un diezmo al planeta base.
- ω_{NS-DIS}: peso del número de naves alojadas en un determinado planeta y la distancia entre dicho planeta y el planeta base, usado en la función de elección del planeta objetivo. Se dispone de un único peso para ambos parámetros ya que estos se encuentran multiplicados en la función de selección.
- ω_{GR}: peso de la tasa de crecimiento en la función de selección del planeta objetivo.
- $pool_{perc}$: proporción de naves extra que se añaden a cada flota enviada desde el planeta base.
- $support_{perc}$: porcentaje de naves extra que se añaden a cada flota enviada desde una colonia a un planeta objetivo.
- $support_{prob}$: probabilidad de enviar una flota extra desde una colonia al planeta objetivo.

Cada parámetro toma valores pertenecientes a rangos diferentes dependiendo de su significado, magnitud e importancia en la IA. Estos valores son empleados en las expresiones que el bot utiliza para tomar las decisiones. Por ejemplo, la función de elección del planeta objetivo se basa en la siguiente puntuación para elegir dicho planeta:

$$Score(p) = \frac{p.NumStarships \cdot \omega_{NS-DIS} \cdot Dist(base, p)}{1 + p.GrowthRate \cdot \omega_{GR}} \quad (1)$$

Este bot ya ofrece un comportamiento mucho más complejo que GoogleBot, y es capaz de vencerle en la gran mayoría de mapas. Sin embargo, en general, AresBot requiere una gran cantidad de turnos para vencer, reflejando que un bot que hubiese tenido una mejor estrategia o más rápida, hubiese ganado casi con total seguridad a AresBot. Es por ello, que se decidió optimizar este conjunto de parámetros, al considerarse inviable (debido a la limitación de tiempo) añadir más reglas al conjunto que define el comportamiento del bot.

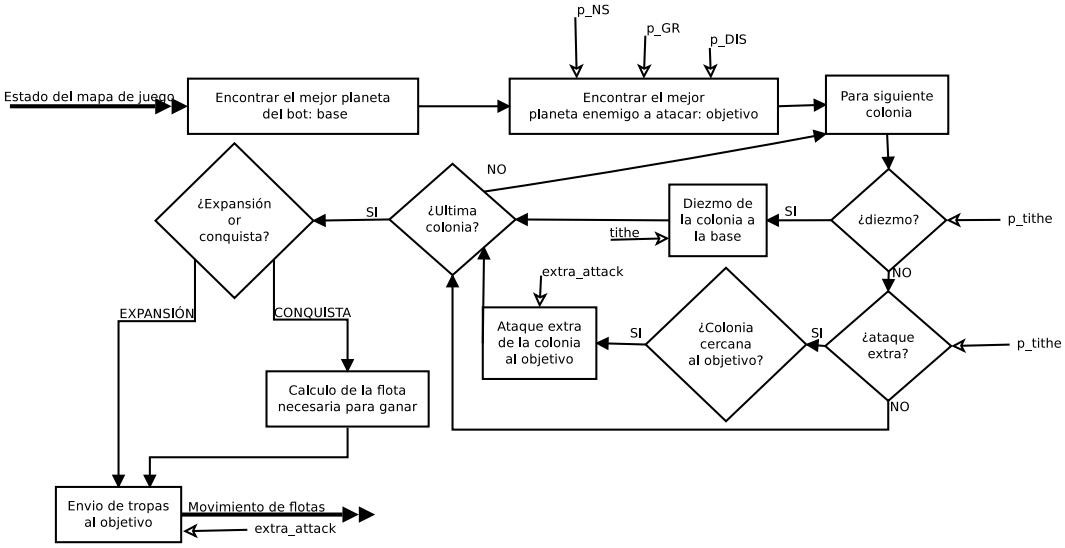

Fig. 2. Diagrama de estados que rigen el comportamiento de AresBot.

C. GeneBot

La principal idea de este artículo es la de realizar una optimización de parámetros off-line mediante la aplicación de un Algoritmo Genético (AG) para el conjunto de parámetros que determinan el comportamiento de AresBot (tal y como se explica en la Subsección IV-B). A dicho bot optimizado, lo hemos denominado *GeneBot*.

El algoritmo de optimización, no forma parte del bot presentado en la competición, sino que una vez encontrados un conjunto de parámetros óptimos estos fueron implementados en el bot subido al sitio del Google AI Challenge.

El AG propuesto, considera codificación real para representar los parámetros en un array y sigue un esquema generacional [16] con elitismo (los mejores bots sobreviven siempre).

Los operadores genéticos utilizados son un operador de cruce *BLX-*α [7] (con α igual a 0.5) muy común en este tipo de codificación de cromosomas para mantener la diversidad, y un operador de mutación que cambia el valor de un gen al azar sumando o restando una cantidad aleatoria en el intervalo $[0, 1]$. Las tasas/probabilidades de aplicación han sido establecidas siguiendo lo habitual en la literatura y después ajustados mediante experimentación sistemática.

El mecanismo de selección implementa un *torneo a 2* en donde dos individuos de la población son elegidos aleatoriamente como padres de un individuo de la próxima generación. También se consideró la selección mediante ruleta de probabilidades, pero nos decantamos por el torneo ya que supone una menor presión selectiva.

El elitismo se ha implementado mediante la sustitución de individuos al azar en la población por los mejores de la actual generación.

La evaluación de un individuo se realiza mediante el establecimiento de los valores correspondientes en los cromosomas como los parámetros de comportamiento de GeneBot y haciendo que este último se enfrente contra GoogleBot en un conjunto de cinco mapas. Dichos mapas han sido elegidos intentando abordar un espectro de posibilidades de distribución lo más amplio posible. Se consideró, que si un bot era capaz de Ganar a GoogleBot en los cinco mapas o lo que es lo mismo en cinco situaciones representativas distintas, y además, el número de pseudo-turnos empleados para ello era mínimo, existía una alta probabilidad de tener también éxito en la mayoría de batallas reales.

El rendimiento de los bots en cada generación, viene determinado por dos valores: el primero es el número de pseudo-turnos empleados para ganar y el segundo es el número de batallas que el bot ha perdido. Siendo más relevante un bot que haya ganado todos sus enfrentamientos que uno que haya tardado menos turnos, pero haya perdido alguno de los suyos.

Un enfoque multi-objetivo podría ser, en principio, una buena aproximación, sin embargo se considera que el punto más importante es ganar el mayor número de partidas (todas en realidad), y una vez hecho esto, minimizar el número de pseudo-turnos requeridos para ello. Esta forma de puntuación se puede ver como una estrategia de implementación de un problema con restricciones: minimizar el número de pseudo-turnos necesarios para ganar *dado que* el individuo es capaz de ganar cada enfrentamiento.

De forma que el *fitness* asociado a un individuo (bot en este caso) sería el mínimo número de pseudo-turnos agregados que se han necesitado para ganar en los cinco mapas representativos.

El código fuente de estos bots se encuentra en: https://forja.rediris.es/svn/geneura/Google-Ai2010.

V. Experimentos y Resultados

Con el objetivo de probar el AG propuesto en la Sección IV-C, se han realizado varios estudios y experimentos. En primer lugar se muestran (Tabla I) los valores de los parámetros considerados para la ejecución del algoritmo. Éstos han sido obtenidos de forma heurística, partiendo de valores considerados estándar en muchos AGs.

Número de Generaciones	100
Número de Individuos	200
Probabilidad de Cruce	0,6
α	0,5
Probabilidad de Mutación	0,02
Política Reemplazo	Elitismo de 2 indivs.

TABLA I
Parámetros utilizados en el algoritmo genético de GeneBot.

A. Optimización de Parámetros

Se han realizado 15 ejecuciones del AG (cada una de ellas de unos 2 días, dado que la evaluación de un individuo consume alrededor de 40 segundos), obteniendo bots cuyos mejores fitness se pueden ver en la Figura 3. Los valores del cromosoma (parámetros evolucionados de AresBot) del *mejor bot* de todas las ejecuciones (perteneciente a la ejecución 8), denominado GeneBot, se muestran en la Tabla II.

Cabe preguntarse si dicho bot es realmente mejor que los obtenidos en el resto de ejecuciones y obviamente, sí resulta un bot mejor que AresBot a la hora de combatir en batallas 'reales'.

Para comprobar estos puntos vamos a enfrentar al mejor bot de cada ejecución contra AresBot en un conjunto de 100 mapas, que proporcionaba Google para probar los bots desarrollados para su AI Challenge 2010, para ver si realmente mejora el comportamiento de AresBot y cual de ellos obtiene los mejores resultados. Dichos resultados los vemos recogidos en la Figura 4 en donde nuevamente observamos como el bot que consigue un mayor porcentaje de victorias frente a AresBot es el bot obtenido en la ejecución 8, GeneBot.

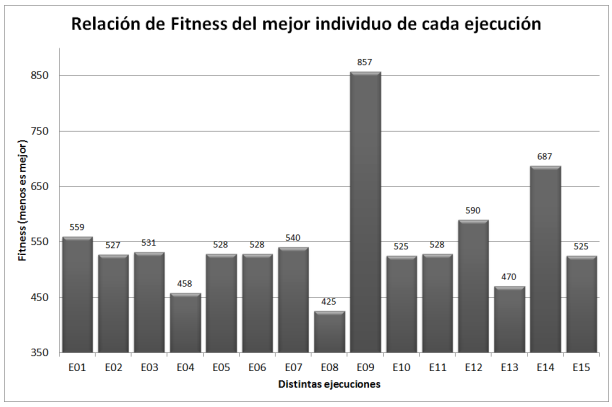

Fig. 3. Fitness de los mejores individuos de cada ejecución.

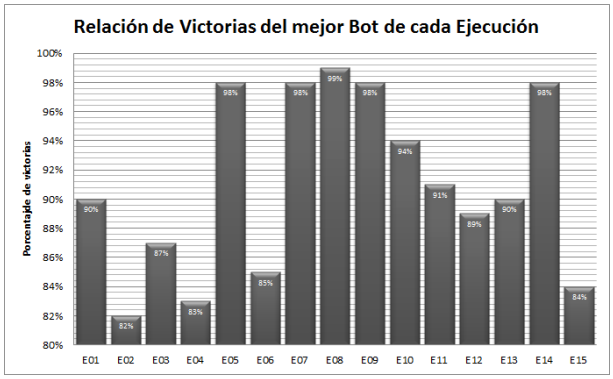

Fig. 4. Porcentaje de victorias de los mejores vs AresBot.

B. Estudio del Fitness Ruidoso

La definición de la función de fitness en todo AG es clave en su éxito, ya que debe modelizar correctamente la bondad de cada individuo. En un entorno pseudo-estocástico (la victoria o derrota depende en gran medida de las acciones del rival) como en el que se encuadra este problema y, por tanto, las evaluaciones de los individuos, habría que comprobar si la función de fitness se comporta de forma estable, es decir, si se garantiza que el fitness obtenido es representativo del individuo o es fruto de una casualidad (o cúmulo de casualidades).

Para paliar este comportamiento azaroso se ha implementado una re-evaluación de los individuos elitistas que sobreviven de una generación a otra, forzando así que los bots sean constantemente probados en combate, así como una función de fitness en la que realizan varios enfrentamientos en campos de batalla muy diferentes entre sí (y representativos de muchas de las posibilidades 'reales').

De igual manera se ha realizado un estudio de tendencia y estabilidad del fitness. En primer lugar, se analizará la tendencia o evolución del fitness durante las generaciones que se desarrolla el algoritmo. Ésta se puede ver en la Figura 5. En dicha figura se puede comprobar la naturaleza *ruidosa* del fitness, pues el fitness del mejor individuo en cada generación, e incluso del mejor absoluto, no siempre decrece o se mantiene, como es habitual en la gran mayoría de problemas.

Para analizar la estabilidad del fitness (ver Figura 6) se muestran los resultados en la evaluación de dos individuos, uno muy prometedor (fitness bajo) y otro poco prometedor (fitness alto). Como se puede ver ambos se mantienen bastante estables en sus rangos de fitness y en relación al número de victorias y derrotas que obtienen respectivamente.

C. Enfrentamientos entre Bots

Por último, vamos a enfrentar a los mejores bots de cada ejecución entre si, con el fin demostrar que el hecho de tener un mejor fitness, representa a un mejor bot, y que incluso esta tendencia se puede observar cuando existe poca diferencia entre los fitness

	$tithe_{perc}$	$tithe_{prob}$	ω_{NS-DIS}	ω_{GR}	$pool_{perc}$	$support_{perc}$	$support_{prob}$
AresBot	0,1	0,5	1	1	0,25	0,5	0,9
GeneBot	0,01799	0,00823	0,50954	0,23273	0,73321	0,58946	0,97405
Media	0,17386	0,09702	0,47252	0,36409	0,65732	0,59987	0.59987

TABLA II

VALORES INICIALES DEL BOT ORIGINAL (ARESBOT) Y VALORES OPTIMIZADOS DEL MEJOR BOT OBTENIDO POR EL AG (EJECUCIÓN 8), LLAMADO GENEBOT, ASÍ COMO LOS VALORES MEDIOS OBTENIDOS POR LOS 15 MEJORES.

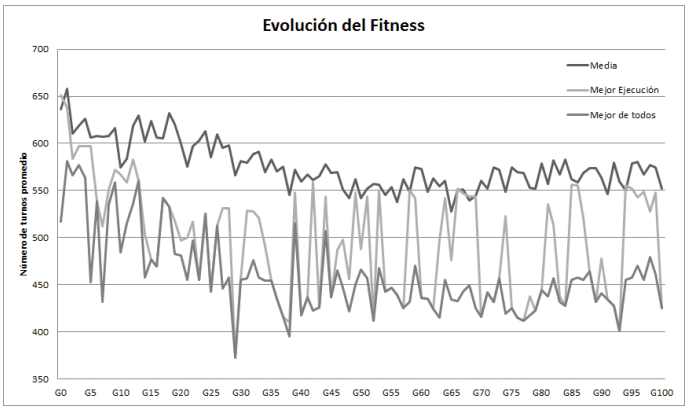

Fig. 5. Evolución del fitness durante las 100 generaciones.

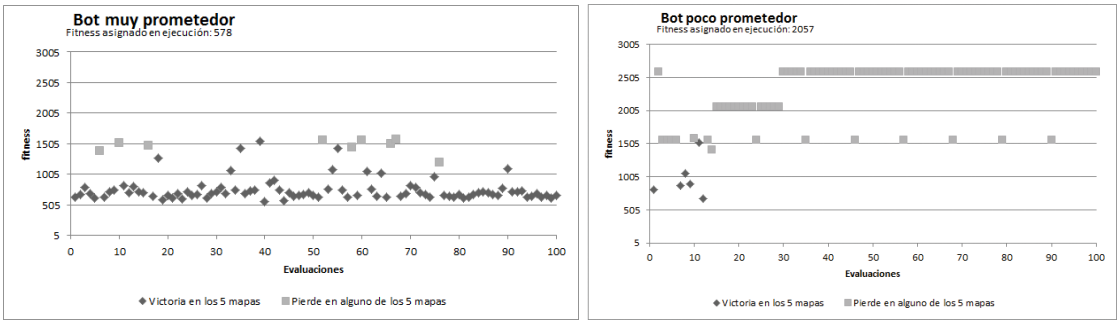

Fig. 6. Estabilidad del fitness en dos individuos diferentes (uno prometedor y otro poco prometedor) en varias evaluaciones.

de los bots que se enfrentan. Estos enfrentamientos se realizan en los 5 mapas representativos. Los resultados se recogen en la Figura 7, en la que se ha marcado como victorioso al bot que gane 3 de los 5 combates.

VI. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se ha presentado un Algoritmo Genético (AG) para la optimización de un conjunto de parámetros que determinan el comportamiento de un agente autónomo (bot), que realiza combates en el juego de estrategia Planet Wars (juego utilizado en el Google AI Challenge 2010). Dicho algoritmo optimizará el comportamiento 'predefinido' de forma heurística de un bot que hemos llamado AresBot, obteniendo el bot bautizado como GeneBot.

En el proceso de la evolución se ha considerado una función de fitness consistente en valorar el número de pseudo-turnos (de 1 segundo por restricciones del juego) empleados en ganar una serie de combates contra un bot estándar del juego. El hecho de realizar varios enfrentamientos en cada evaluación (considerando además 5 mapas que representan situaciones muy dispares), junto con el hecho de re-evaluar en cada generación incluso al mejor individuo hasta el momento, se han realizado para tratar la 'naturaleza ruidosa' de dicho fitness (el resultado de un enfrentamiento puede variar para un mismo individuo en función de las acciones del rival).

Los experimentos llevados a cabo demuestran que los bots (sus parámetros de comportamiento) obtenidos tras la evolución son mucho mejores que el bot definido de forma heurística, y completamente superiores al bot estándar. Además, dichos experimentos demuestran que el tratamiento que se ha dado al 'fitness ruidoso' del problema ha sido correcto, pues a pesar de él, los bots prometedores dentro de una ejecución obtienen siempre una tasa de victorias muy superior al resto, al igual que ocurre con los bots netamente peores, que no consiguen apenas victorias durante su evaluación.

Como apunte decir que GeneBot terminó en el

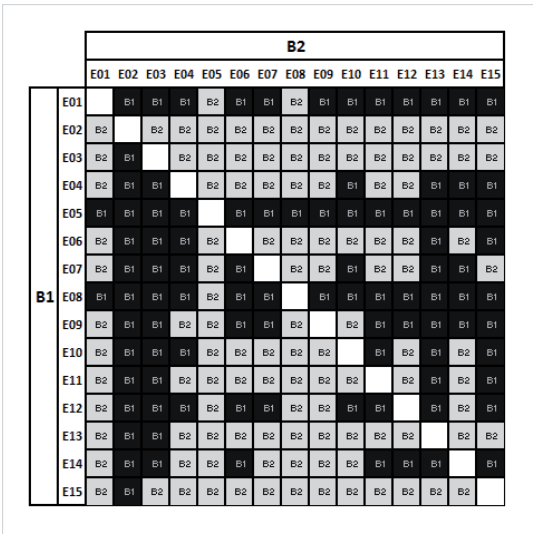

Fig. 7.
Resultado de los enfrentamientos entre los mejores bots de cada ejecución en 5 mapas.

puesto 1454 de la competición, entre el 30 % de los mejores, y mucho mejor que el original AresBot que se situó en una posición mayor de 2000.

Este estudio es relativamente reciente para los autores, por lo que hay muchos frentes posibles de trabajo abiertos. Cuestiones como la co-evolución de individuos, el uso de Programación Genética para definir el comportamiento del bot en base a reglas optimizadas, o la consideración de un algoritmo multiobjetivo para abordar distintos criterios en la optimización, son algunas de las líneas que se desean estudiar en próximos trabajos.

Agradecimientos

Este trabajo se ha llevado a cabo dentro de los proyectos TIN2011-28627-C04-02, Junta de Andalucía P08-TIC-3903, UGR PR-PP2011-5, y CEI BioTIC GENIL (CEB09-0010) del Programa CEI del MICINN (PYR-2010-13 y PYR-2010-29).

Referencias

[1] P. Avery and S. Louis, "Coevolving team tactics for a real-time strategy game," in *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, 2010.

[2] N. Beume *et al.*, "Intelligent anti-grouping in real-time strategy games," in *International Symposium on Computational Intelligence in Games*, I. Press, Ed., Perth, Australia, 2008, pp. 63–70.

[3] D.E. Goldberg, B. Korb, K. Deb, *Messy genetic algorithms: motivation, analysis, and first results*,Complex Systems, Vol. 3(5), pp. 493–530, 1989.

[4] A. I. Esparcia-Alcázar, A. I. M. García, A. Mora, J. J. M. Guervós, and P. García-Sánchez, "Controlling bots in a first person shooter game using genetic algorithms," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[5] W. Falke-II and P. Ross, "Dynamic Strategies in a Real-Time Strategy Game," *E. Cantu-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 1920-1921, Springer-Verlag Berlin Heidelberg*, 2003.

[6] Google, "Google AI Challenge 2010," *http://ai-contest.com*, 2010.

[7] F. Herrera, M. Lozano, and A. M. Sánchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study," *International Journal of Intelligent Systems*, vol. 18, pp. 309–338, 2003.

[8] J. Hagelbäck and S. J. Johansson, "A multi-agent potential field-based bot for a full RTS game scenario," in *AIIDE*, C. Darken and G. M. Youngblood, Eds. The AAAI Press, 2009.

[9] S.-H. Jang, J.-W. Yoon, and S.-B. Cho, "Optimal strategy selection of non-player character on real time strategy game using a speciated evolutionary algorithm," in *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09)*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 75–79.

[10] D. Keaveney and C. O´Riordan, "Evolving robust strategies for an abstract real-time strategy game," in *International Symposium on Computational Intelligence in Games*, I. Press, Ed., Milano. Italy, 2009, pp. 371–378.

[11] J. E. Laird, "Using a computer game to develop advanced AI," *Computer*, pp. 70–75, 2001.

[12] L. Lidén, "Artificial stupidity: The art of intentional mistakes," in *AI Game Programming Wisdom 2*. Charles River Media, INC., 2004, pp. 41–48.

[13] D. Livingstone, "Coevolution in hierarchical ai for strategy games," in *IEEE Symposium on Computational Intelligence and Games (CIG05)*. Essex University, Colchester, Essex, UK: IEEE, 2005.

[14] S. Lucas, "Computational intelligence and games: Challenges and opportunities," *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 45–57, 2008.

[15] E. Martín, M. Martínez, G. Recio, and Y. Saez, "Pac-mant: Optimization based on ant colonies applied to developing an agent for ms. pac-man," in *Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On*, G.Ñ. Yannakakis and J. Togelius, Eds., August 2010, pp. 458–464.

[16] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Springer, 1996.

[17] C. Miles and S. J. Louis, "Co-evolving real-time strategy game playing influence map trees with genetic algorithms," in *International Congress on Evolutionary Computation*, I. Press, Ed., Portland, Oregon, 2006.

[18] A. M. Mora, M. Ángel Moreno, J. J. Merelo, P. A. Castillo, M. I. G. Arenas, and J. L. J. Laredo, "Evolving the cooperative behaviour in Unreal^{TM} bots," in *Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On*, G.Ñ. Yannakakis and J. Togelius, Eds., August 2010, pp. 241–248.

[19] E. Onieva, D. A. Pelta, J. Alonso, V. Milanés, and J. Pérez, "A modular parametric architecture for the torcs racing engine," in *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09)*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 256–262.

[20] S. Ontanon, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science, R. Weber and M. Richter, Eds. Springer Berlin / Heidelberg, 2007, vol. 4626, pp. 164–178.

[21] M. Ponsen, H. Munoz-Avila, P. Spronck, and D. W. Aha, "Automatically generating game tactics through evolutionary learning," *AI Magazine*, vol. 27, no. 3, pp. 75–84, 2006.

[22] R. Small and C. Bates-Congdon, "Agent Smith: Towards an evolutionary rule-based agent for interactive dynamic games," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, May 2009, pp. 660–666.

[23] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Improving opponent intelligence through offline evolutionary learning," *International Journal of Intelligent Games & Simulation*, vol. 2, no. 1, pp. 20–27, February 2003.

[24] J. Togelius, S. Karakovskiy, J. Koutnik, and J. Schmidhuber, "Super mario evolution," in *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09)*. Piscataway, NJ, USA: IEEE Press, 2009.

[25] Wikipedia, "Computer Game Bot — Wikipedia, The Free Encyclopedia", `http://en.wikipedia.org/wiki/Computer_game_bot`

[26] Wikipedia, "Galcon — Wikipedia, The Free Encyclopedia", `http://en.wikipedia.org/w/index.php?title=Galcon&oldid=399245028`