Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Proyecto de fin de Carrera
Ingeniería en Informática
Curso 2011/2012

Diseño de estrategias multiagente para el control de equipos de bots en el entorno del videojuego Unreal Tournament 2004

Carlos Sánchez Serrano

Director: Manuel González Bedia
Co-Director: Francisco Serón Arbeloa

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Abril de 2012

Diseño de estrategias multiagente para el control de equipos de bots en el entorno del videojuego Unreal Tournament 2004

RESUMEN

En el presente proyecto se pretende llevar a cabo una serie de simulaciones en el contexto de los videojuegos en primera persona, que demuestre la capacidad de los sistemas multiagente a la hora de resolver problemas de exploración y búsqueda en entornos. Emplearemos técnicas basadas en algoritmos bioinspirados, tales como el Swarming, y la organización en roles de nuestros agentes (bots) en sistemas distribuidos, estudiando las posibles interacciones y dependencias entre ellos. Se analizará el contexto en el que nos encontramos actualmente, el estado del arte de la inteligencia artificial en el mundo de los videojuegos y las diferentes metodologías y técnicas empleadas para la realización de nuestras simulaciones.

Nuestros experimentos estarán enmarcados dentro del dilema "exploration vs exploitation", una paradoja presente en todos los sistemas que pueden adaptarse y aprender [Holland, 1992], donde se intenta buscar el equilibrio entre dos tipos de comportamiento que afectan directamente a la eficiencia a la hora de resolver una tarea. Estrategias de computación evolutiva por medio de algoritmos genéticos, nos ayudarán a crear nuevas generaciones de agentes que nos permitirán ajustar y optimizar nuestros sistemas.

El videojuego para el cual se programarán nuestros bots cooperativos será el Unreal tournament 2004 (UT2004). Se trata de un videojuego tipo shooter en primera persona (FPS, Ver glosario) donde los objetivos pueden centrarse en enfrentarse a un enemigo o capturar la bandera enemiga. La implementación la llevaremos acabo a través del conjunto de librerías Pogamut [Gemrot et al., 2009], una plataforma de código abierto usada para el rápido desarrollo de comportamientos en agentes virtuales incrustados en un entorno 3D del videojuego Unreal Tournament 2004, que nos permite codificar los agentes mediante el uso de Java.

Por otro lado, se pretende abrir una nueva línea de investigación desde la Universidad de Zaragoza, que gire entorno a la programación de agentes inteligentes sobre la plataforma UT2004 y Pogamut. Nuestro marco formal previo a los experimentos y las simulaciones en sí mismas, pretenden explorar distintas estrategias multiagente para resolver problemas que puedan ser útiles para trabajos futuros.

Además, al tratarse de un trabajo pionero, el objetivo previo a la realización de este proyecto consistirá en estudiar las plataformas empleadas, analizar sus ventajas y limitaciones, así como realizar un manual adecuado para programadores, dado que Pogamut al principio de este proyecto no contaba con uno. Con ello se pretende sentar las bases para futuros proyectos en el área de IA en videojuegos utilizando Pogamut y Unreal Tournament 2004.

Agradecimientos

Quiero agradecer a todas las personas que me han apoyado durante el trascurso de la carrera, ya sea de manera directa o indirecta.

A mis padres, Carlos y Pilar, que tienen más ganas que yo de que esto termine y a mi hermano Pablo, el cual espero que se quede solo con lo bueno de mí. También para todos los amigos que me han apoyado en muchos aspectos.

A mis compañeros de PFC, Jol, Sergio y Ángel, por su lucha aliada contra Pogamut y su filosofía de compartir conocimientos.

A mis tutores de proyecto, Paco y Manolo, por brindarme la oportunidad de investigar sobre lo que me gusta y por haberme ayudado en todo lo posible. No quiero dejarme de nombrar al maestro de la IA académica, al cual no le ha importado ayudarme a altas horas de la madrugada para mejorar mi proyecto. También a Germán, el gurú probabilístico, que nos ha ayudado todo lo que ha estado en su mano.

Por último, se lo quiero agradecer a Claudia por su apoyo constante en todos los ámbitos, porque sin ella no estaría escribiendo este texto.

Índice general

Índice de figuras

Índice de tablas

Parte I

Memoria

Capítulo 1

Introducción

1.1. Motivación y objetivos

En el año 2011, el mercado de los videojuegos superó a la industria del cine y de la música juntas, generando unos ingresos superiores a los 23.000 millones de euros[1]. Con un coste de la potencia computacional cada vez menor, los gráficos y la física de los juegos son cada vez más realistas. Sin embargo, la inteligencia artificial (IA) empleada en los videojuegos parece no crecer al mismo ritmo.

Actualmente, los desarrolladores empiezan a reconocer la necesidad de una mejora de la IA en los videojuegos, de forma que el comportamiento de los NPC's (Non-Player Characters, ver glosario) sea mas impredecible y más parecido al de un jugador humano [J. Laird, 2000]. No obstante, todavía existe un fuerte contraste entre las técnicas de IA usadas en la industria del videojuego (entre las que destacan las máquinas de estados finitas, árboles de comportamiento y algoritmos de planificación) y otras utilizadas fundamentalmente en la investigación académica (más orientadas a metodos de aprendizaje automático, redes neuronales y algoritmos genéticos). Creemos que en el tránsito y adaptación de modelos académicos a entornos comerciales puede estar la clave de la nueva generación de los videojuegos del futuro.

El principal objetivo del presente proyecto es llevar a cabo una serie de experimentos, en el contexto de los videojuegos en primera persona, que demuestre la capacidad de los sistemas multi-agente a la hora de resolver problemas de exploración de entornos. Emplearemos técnicas basadas en algoritmos bioinspirados y la organización en roles de nuestros agentes (bots) en sistemas distribuidos, estudiando las posibles interacciones y dependencias entre ellos. Nuestros experimentos estarán enmarcados dentro del dilema "exploration vs exploitation", una paradoja presente en todos los sistemas que pueden adaptarse y aprender [Holland, 1992], donde se intenta buscar el equilibrio entre dos tipos de comportamiento que afectan directamente a la eficiencia a la hora de resolver una tarea. Este equilibrio estará entre actuar inmediatamente basándose en la mejor información que disponemos actualmente o esperar a conseguir más información, la cual puede permitirnos posteriormente obtener un mejor rendimiento. Obtener más información conlleva una perdida de rendimiento, mientras que la explotación de los mejores datos que tenemos conlleva el riesgo de perpetuar un error [Holland, 1992]. Existen algunos modelos exitosos en el tratamiento de este dilema; en particular, nos centraremos en analizar el trabajo de [Vergassola et al, 2008], para definir un modelo propio adaptado al entorno de trabajo de este proyecto.

El videojuego para el cual se programarán nuestros bots cooperativos será el Unreal tournament 2004 (UT2004). Se trata de un videojuego tipo shooter en primera persona (FPS, first-person

[1] Según "The Entertainment Software Association" - http://www.theesa.com

shooter) donde los objetivos van desde acabar con el enemigo hasta capturar la bandera enemiga. Unreal Tournament 2004 es la herramienta empleada en el concurso a nivel mundial 2kBotPrize[2], celebrado todos los años, donde investigadores de todo el mundo presentan sus bots con el objetivo de superar una adaptación del test de Turing. Varios jueces tendrán que evaluar y votar durante varias partidas el comportamiento de los bots programados, obteniendo éstos una puntuación que corresponde a su grado de humanidad. Además, este videojuego cuenta con la plataforma Pogamut [Gemrot et al., 2009], desarrollada en la Universidad de Praga y que se encuentra actualmente en su versión 3.2, la cual nos facilita un conjunto de librerías para implementar nuestros bots en el lenguaje de programación Java.

Otro objetivo del proyecto es abrir una nueva línea de investigación desde la Universidad de Zaragoza que gire entorno a la programación de agentes inteligentes en la plataforma UT2004. Para ello, primero deberemos demostrar la capacidad y validez de Pogamut a la hora realizar este tipo de estudios, comprobando que nos permita realizar las simulaciones de nuestros experimentos con las menores limitaciones posibles. Al tratarse de un trabajo pionero, será necesaria una fase previa a la realización de los experimentos, que consistirá en la formación en dicha plataforma. Nuestro marco formal previo a los experimentos y las simulaciones en sí, pretenden explorar distintas estrategias multiagente para resolver problemas que puedan ser útiles para trabajos futuros que giren en torno a los videojuegos. Además se realizará un manual adecuado para programadores, dado que Pogamut al principio de este proyecto no contaba con uno. Con ello se pretende sentar las bases para futuros proyectos en el área de IA en videojuegos utilizando Pogamut y Unreal Tournament 2004.

1.2. Contexto

1.2.1. ¿Pueden pensar las máquinas?

Con esta sugerente pregunta comenzaba el artículo "Computing Machinery and Intelligence" publicado por Alan Turing en la revista de filosofía británica Mind (1950). Aquí se trató por primera vez la cuestión de la posible existencia de una inteligencia artificial y propuso el famoso test de Turing [Mind, 1950], considerando que si una máquina se comporta en todos los aspectos como inteligente, entonces debe ser inteligente. Mediante dicho test, Turing introduce un criterio operacional para dirimir si el comportamiento de un artefacto debería ser considerado como inteligente. En principio parecería obvio que este atributo solo se pueda asociar a los seres humanos, que durante millones de años han desarrollado las estructuras neuronales que conforman sus complejos cerebros. Sin embargo, la respuesta a esta pregunta no es sencilla. Dependerá, como bien dijo Turing, de qué entendamos por pensar.

Según la Real Academia Española de la Lengua, "pensar" supone *"el acto de examinar con cuidado algo para formar dictamen"*. Una definición de este tipo deja una puerta abierta a la existencia de máquinas inteligentes.

La intención del test de Turing es la de corroborar la existencia de inteligencia en las máquinas. La prueba está estructurada de forma que un jurado situado en una habitación, se encargaba de formular preguntas a dos competidores (una máquina y un ser humano) situados en otras habitaciones y, basándose en sus respuestas, debía decidir cuál era la máquina y cuál el ser humano. De este modo, se consideraría que el test había sido superado si el jurado no fuese capaz de descubrir la identidad real de cada uno de los participantes. Desde entonces, han surgido numerosas variantes de este test que persiguen un mismo objetivo, pero con aplicación en diversos campos de investigación. Hasta el día de hoy, ninguna máquina ha sido capaz de superar el test con éxito[3].

En el mundo de los videojuegos, que los NPC's superaran el test de turing, supondría que el usuario no sabría si esta jugando contra una persona o un bot controlado por el computador.

[2] http://botprize.org
[3] http://www.loebner.net/Prizef/loebner-prize.html

Figura 1.1: Explicación gráfica del test de Turing

1.2.2. Inteligencia Artificial y Videojuegos

Durante la década de los 90 comenzaron a utilizarse, aunque de forma muy rudimentaria, técnicas vinculadas a la Inteligencia Artificial (IA) con el objetivo de desarrollar videojuegos mucho más complejos que los que había hasta entonces. Esta necesidad venía acuciada por los juegos enormemente predecibles que había en aquella época, en los que las acciones realizadas por los jugadores no controlados por el usuario (non-playing characters, NPC's), se repetían con cierta asiduidad. Videojuegos como Herzog Zwei[4] o Dune II[5] comenzaban a aplicar máquinas de estados para poder desarrollar estrategias en tiempo real. Del mismo modo, en 1996 el videojuego Battlecruiser 3000AD[6] utilizó por primera vez redes de neuronas artificiales.

En este contexto, años más tarde nace un videojuego que merece una mención especial. Este videojuego es Golden Eye[7], basado en las aventuras del personaje cinematográfico James Bond, al convertirse en el primer juego de tipo FPS que aplicaba técnicas de Inteligencia Artificial durante el juego. De este modo, los NPC's reaccionaban convenientemente al movimiento del jugador controlado por el usuario. Aún así, este juego presentaba un gran inconveniente: los enemigos del usuario sabían en todo momento dónde se encontraba éste, incluso si aún no lo habían visto, con lo que era imposible esconderse para desarrollar acciones más reales de comportamiento.

En actualidad, se ha producido un incremento significativo en la producción de videojuegos. El mercado de los videojuegos ha superado a la industria del cine y de la música juntas, con unas ventas superiores a los 23.000 millones de euros. Con un coste cada vez menor de la potencia computacional, los gráficos y la física de los juegos son cada vez más realistas.

Sin embargo, estos no son los únicos elementos que hacen que un juego de éxito. El comportamiento de los NPC's es incluso más importante a la hora de hacer un juego realista y entretenido para el usuario. La industria parece no ser consciente del problema y muestra reticencias a la hora de cambiar las técnicas de IA tradicional utilizadas, tales como máquinas de estados finitas, árboles de comportamiento y algoritmos de planificación. Incluso en los juegos que han visto la luz recientemente, la comunidad de jugadores parece no estar satisfecha con el IA de los juegos.

El objetivo de toda IA aplicada a videojuegos debería ser que los bots mostraran un comportamiento humano y por tanto realista. Las expectativas de los jugadores deben ser cumplidas. Dichas expectativas se pueden clasificar en tres clases [J. Laird, 2000]:

1. Los jugadores esperan nuevas situaciones. Las situaciones repetitivas hacen que los jugadores se desinteresen. Si hacemos que los bots tengan un comportamiento impredecible a la vez que coherente, conseguiremos crear nuevas situaciones.

[4] http://en.wikipedia.org/wiki/Herzog_Zwei
[5] http://duneii.com/
[6] http://www.3000ad.com/
[7] http://en.wikipedia.org/wiki/GoldenEye_007

2. También se espera una total interactividad. Los jugadores quieren que los bots reaccionen ante ellos y sean capaces de modificar su comportamiento dependiendo de las acciones del usuario. También deben ser conscientes del entorno y reaccionar ante él.

3. Es deseable un alto grado de desafío. Dentro de la comunidad de jugadores, especialmente en los más avanzados que llevan jugando durante años, necesitan nuevos retos en el juego para no perder el interés en él. Los bots deben hacerlo bien, pero sin llegar al extremo de tener el control y conocimiento sobre todas las cosas, al igual que un jugador humano.

Existe un problema con los videojuegos comerciales mejor desarrollados y es que, en el mejor de los casos, llegan a desvelar en qué aspectos aplican técnicas de Inteligencia Artificial. No obstante no se proporciona información detallada respecto a la técnica en cuestión, por lo que únicamente se pueden emitir conjeturas respecto a cuáles son las que, con mayor probabilidad, se han aplicado en cada caso. Sin embargo, existen juegos que a pesar de haber tenido una gran aceptación popular, mantienen su código abierto a todos los usuarios o, al menos, facilitan herramientas para poder personalizarlos, de manera que sea mucho más sencillo comprender el funcionamiento interno del mismo. Este es el caso de Unreal Tournament 2004, usado en el presente proyecto.

1.3. Contenido de la Memoria

La estructura de este documento esta dividida en los siguientes capítulos:

1. Introducción: Se establecen los objetivos, el contexto en el que se encuadra el trabajo y la planificación que se ha llevado a cabo para realizar este proyecto.

2. Estado del arte: Se examinan las diferentes estrategias que se han llevado a cabo en los últimos años en el diseño de comportamientos colectivos.

3. Marco formal y metodología: Se detalla el marco formal donde se encuadran nuestros experimentos así como las técnicas que se han empleado para la realización de este proyecto, tales como los comportamientos de enjambre y los algoritmos genéticos.

4. Herramientas utilizadas: En este capitulo se detallan las herramientas que nos han permitido realizar nuestros experimentos.

5. Experimento 1 - siSosiG Bot: Se plantea un experimento basado en un comportamiento de roles implícito, donde se detallan sus objetivos, su implementación y los resultados obtenidos junto a las conclusiones sacadas.

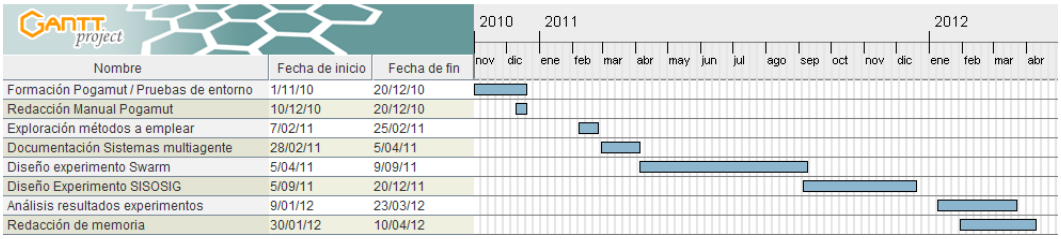
6. Experimento 2 - PathWalker Bot: En este capítulo analizamos el experimento de nuestros bots Pathwalkers, donde un equipo con comportamiento de enjambre es capaz de resolver una tarea. Para llevar a cabo su análisis, se presentan los objetivos, implementación y resultados al igual que en el experimento previo.

7. Conclusiones: Conclusión general sobre el desarrollo del proyecto, posibles líneas futuras y valoración personal del proyecto.

8. Anexos: Información adicional del proyecto, referenciada desde apartados anteriores.

1.4. Planificación

Durante los meses de duración del proyecto, se han realizado labores de documentación y formación en las plataformas usadas, se ha redactado el manual de Pogamut, se han implementado y ejecutado los experimentos, así como se ha llevado a cabo un análisis de resultados y se han sacado conclusiones de los mismos. De manera paralela se ha escrito el presente documento. El diagrama de Gantt correspondiente a la realización del proyecto se muestra en la figura 1.2.



Nombre	Fecha de inicio	Fecha de fin
Formación Pogamut / Pruebas de entorno	1/11/10	20/12/10
Redacción Manual Pogamut	10/12/10	20/12/10
Exploración métodos a emplear	7/02/11	25/02/11
Documentación Sistemas multiagente	28/02/11	5/04/11
Diseño experimento Swarm	5/04/11	9/09/11
Diseño Experimento SISOSIG	5/09/11	20/12/11
Análisis resultados experimentos	9/01/12	23/03/12
Redacción de memoria	30/01/12	10/04/12

Figura 1.2: Diagrama Gantt del proyecto

Capítulo 2

Estado del arte

Antes de pasar a detallar nuestros experimentos en UT2004, vamos a presentar el estado del que nos ha empujado a llevar a nuestra propuesta de modelo, con objetivo de asentar las bases de la problemática a la que nos enfrentamos y facilitar el trabajo futuro en esta línea de trabajo.

2.1. Sistemas multiagente

Existen tareas, independientes del contexto donde nos encontremos, en las que un único agente no es capaz de llevarlas a cabo, o si lo puede, lo hace de manera muy ineficiente y costosa. En esos casos, el trabajo en equipo de varias entidades puede ser una buena opción.

Entramos así, en el contexto de los sistemas multiagente (figura 2.1[1]), que permiten la gestión inteligente de un sistema complejo, coordinando los distintos subsistemas que lo componen e integrando los objetivos particulares de cada subsistema en un objetivo común. Estos sistemas se emplean cuando los problemas son físicamente distribuidos, cuando la solución requiere de experiencia muy heterogénea o cuando el problema a resolver está definido sobre una red de computadores. En realidad, la complejidad de la mayor parte de los problemas que nos encontramos hoy en día es tal que se requiere una solución distribuida, capaz de adaptarse a cambios en la estructura y en el entorno, así como una metodología de desarrollo que permita la construcción de todo un sistema a partir de distintas unidades autónomas [Ferber, 1999].

En general, un sistema multiagente cooperante [Wesson et al., 1988] presentará las siguientes características:

- Estará formado por un conjunto de agentes, cada uno de los cuales mantiene sus propias habilidades: adquisición de datos, comunicación, planificación y actuación.
- El sistema multiagente tiene una misión común. La misión puede descomponerse en diferentes tareas independientes, de forma que se pueden ejecutar en paralelo. El sistema multiagente debe ser capaz de asignar a cada uno de sus componentes una o varias tareas concretas teniendo en cuenta cuál es el objetivo común.
- Cada agente del sistema tiene un conocimiento limitado. Esta limitación puede ser tanto del conocimiento del entorno, como de la misión del grupo, como de las intenciones de los demás agentes a la hora de realizar sus propias tareas.

[1] http://www.codeproject.com/Articles/13544/Agents-and-Multi-agent-Sys

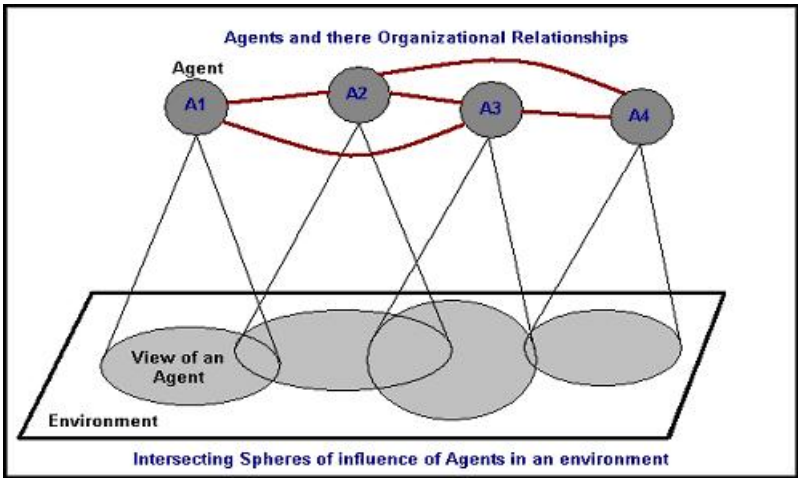


Figura 2.1: Representación de sistemas multiagente

- Cada agente del sistema tiene cierta especialización para realizar determinadas tareas, en función de lo que conoce, la capacidad de proceso y la habilidad requerida.

Ronald Arkin (1998) enumera algunos de los aspectos positivos del trabajo en equipo con varios agentes:

- Mejora en el rendimiento del sistema, ya que si las tareas pueden descomponerse de un modo natural, la estrategia de divide y vencerás es apropiada.
- Se pueden realizar ciertas tareas que serian imposibles para un solo bot; un equipo puede llevar a cabo de modo simultáneo acciones en diferentes localizaciones.
- Presentan una mayor tolerancia a fallos ante un mal funcionamiento de algún bot.

Pero no todo son ventajas, ya que el uso de sistemas multiagente también puede traer consigo algunas consecuencias. Como aspectos negativos, podríamos destacar:

- Interferencia a la hora de realizar la tarea, que Arkin explica con el conocido dicho de que "demasiados cocineros estropean el guiso", dada la mayor probabilidad de choques o bloqueos.
- La necesidad de comunicación, directa o indirecta, con el coste computacional y estructural que ello conlleva.

En el problema de la coordinación entre los agentes es tan importante el proceso de razonamiento interno como el proceso de comunicación. El proceso de razonamiento interno consistirá en la toma de decisiones y en la identificación de la información que se debe compartir. El proceso de comunicación debe modelar cómo y cuándo debe producirse la interacción entre los agentes.

2.1.1. Técnicas clásicas empleadas para el modelado de comportamiento colectivo

Para intentar modelar el comportamiento de equipos de agentes se han empleado diferentes técnicas a lo largo de los años, las cuales podemos englobar en dos tipos principales: algoritmos tipo *flocking* y algoritmos voraces.

Algoritmos tipo flocking

Este tipo de algoritmos han sido empleados para modelar sistemas de vida artificial, que se componen de un conjunto de organismos sintéticos modelados mediante reglas simples de las que emergen comportamientos característicos de los sistemas naturales vivos [Langton, 1989]. La comunidad científica estudia las cualidades inherentes a los seres vivos como son el crecimiento, la reproducción, la percepción del entorno, la respuesta al medio ambiente, la adaptabilidad, el metabolismo, la autonomía, la capacidad de reacción y la evolución con objeto de resolver problemas, utilizando agentes inspirados biológicamente que exhiben un comportamiento colectivo inteligente [Terzopoulos, 1999]. En particular, en el trabajo pionero de Reynolds, se modela el comportamiento de las bandadas, manadas o muchedumbres, utilizando tres reglas sintácticas muy sencillas y logrando la emergencia de comportamientos relacionados con la alineación, no colisión y movimiento de agregación [Bajec et al, 2007].

Sin embargo, en la mayoría de los trabajos de investigación relacionados con el tipo de problemas citados, a cada individuo hay que dotarle de un conocimiento completo de la posición, velocidad y dirección de vuelo de, al menos, un subconjunto de los individuos de la bandada. Por decirlo de una manera más expresiva, dentro de su conciencia del mundo los datos que utiliza son bastante exactos sobre magnitudes normalmente físicas. Ese tipo de requerimientos son antinaturales (un pájaro o un pez tiene una capacidad de percepción limitada e inexacta), y por lo tanto la filosofía del modelo se aleja de los enfoques naturalistas y entra en contradicción con las premisas de qué es un sistema de vida artificial.

Algorimos voraces

En algoritmo un voraz siempre se hace la mejor elección disponible en cada paso (óptimos locales) de la implementación, con la esperanza de que de esta manera se pueda obtener el mejor resultado global. En contraste, métodos como los algoritmos genéticos, discutido abajo, no son voraces; a veces, estos métodos hacen elecciones menos óptimas al principio con la esperanza de que conducirán hacia una solución mejor más adelante.

El uso de algoritmos voraces en contextos de exploración y búsqueda sin modelos basados en la reducción del gradiente de la concentración de un estímulo. Según el tipo de fuente del estímulo, podemos encontrar diferente implementaciones en la literatura científica: Quimiotaxis si el comportamiento esta guiado por un gradiente químico, Fototaxis cuando el tenemos un estímulo luminoso, Fonotaxis cuando la fuente es sonora, etc. Si un agente es modelado mediante este tipo de estrategias tendría un comportamiento descrito por las siguiente acciones:(1) realizaría un movimiento, (2) mediría el cambio en la señal sensorial, (3) si el cambio es positivo seguiría moviéndose en la misma dirección, en caso contrario, cambiaría el sentido de su movimiento. Tal estrategia de búsqueda requiere que la concentración del estímulo sea lo suficientemente alta como para asegurar que la diferencia entre las medidas en posiciones vecinas sea mayor que sus fluctuaciones [Berg, 1993]. Existen diversos trabajos donde se emplean estás técnicas de manera exitosa ([Russel et al, 2003],[Grasso et al, 2000]) pero en todos ellos la relación señal-ruido(SNR) es muy alta para que el rendimiento del agente sea aceptable. Sin embargo, estos métodos de búsqueda no resultan satisfactorios cuando nos encontramos en entornos dinámicos o con incertidumbre.

Desde que [Barlow, 1969] propusiera el principio de codificación eficiente se acepta que para que un sistema procese señales de forma eficaz en entornos con incertidumbre debe aprovecharse de la estructura estadística de la señal de entrada. Es decir, que un agente más que combatir la incertidumbre debería tratar de beneficiarse de ella. Estos comportamientos son observados frecuentemente en organismos vivos y de igual forma constituyen un campo de trabajo en la robótica[Hamza, 2006].

2.1.2. Técnicas empleadas para el modelado de comportamiento colectivo en entornos inciertos

Cuando un agente se encuentra en un entorno dinámico, modelos prediseñados como los citados anteriormente, no proporcionan un comportamiento satisfactorio. Como el agente no posee, en un principio, una estrategia para actuar correctamente, deberá evaluar las diferentes acciones posibles mediante un proceso de ensayo y error [Sutton et al, 1998]. Existen varios métodos que han sido desarrollados para resolver este tipo de tareas: métodos de Montecarlo, actualización de diferencias temporales, trazas de elegibilidad [Sutton et al, 1998],etc.

Todos estos métodos presentan dos aspectos incompatibles: (1) por un lado para maximizar la evaluación los agentes deben seleccionar las acciones que ya conocen, y por otro lado, (2) se deberían probar nuevas acciones para descubrir nuevas posibilidades. Si nos centráramos exclusivamente en el primer aspecto estaríamos explotando solo la información de la que disponemos, sin posibilidad de adaptarnos a nuevas situaciones. En cambio un agente que base su comportamiento solo en el segundo aspecto, ejecutaría acciones sin ningún propósito concreto. Por este motivo, el comportamiento ideal se construiría mediante un equilibrio entre las dos estrategias. Tradicionalmente, la búsqueda del equilibrio entre estos dos factores en entornos cambiantes, recibe el nombre del "dilema de exploración-explotación".

2.2. Dilema de exploración-explotación

El *"dilema de exploración vs explotación"* (exploration vs exploitation, E-E), plantea el problema de encontrar el óptimo global en un espacio que posee óptimos locales. Esta paradoja se produce en todos los sistemas que pueden adaptarse y aprender [Holland, 1992].

La decisión de aprender es fundamentalmente una elección entre actuar inmediantamente basándose en la mejor información que disponemos actualmente o esperar a conseguir más información, la cual puede permitirnos posteriormente obtener un mejor rendimiento. Obtener más información conlleva una perdida de rendimiento, mientras que la explotación de los mejores datos que tenemos conlleva el riesgo de perpetuar un error (Holland, 1992). Si nos decantamos por recavar más información acerca del entorno en el que estamos, estaremos incurriendo en un coste a corto plazo para conseguir una mejor respuesta en un periodo más largo. En cambio, si empleamos la información de la que ya disponemos, obtendremos en principio un mejor rendimiento al obtener los beneficios de actuar ahora. Podríamos definir[2] cada una de las estrategias como:

- Exploración: Incluye estrategias representadas por términos como búsqueda, variación, experimentación, juego, flexibilidad, descubrimiento, innovación.
- Explotación: Sus estrategias van definidas en el sentido de refinamiento, elección, producción, eficiencia, selección, implementación y ejecución.

El equilibrio entre aprendizaje y rendimiento es el objetivo a resolver en un problema donde se nos plantee el dilema E-E.

Este tipo de modelos se han empleado para modelar el comportamiento de agentes con aprendizaje por refuerzo [Thrun, 1992]. Dicho aprendizaje consiste en aprender a decidir antes una situación determinada, qué acción es la más adecuada para logar un determinado objetivo basándose en resultados previos. En este marco podemos distinguir dos facetas, las cuales nos podemos encontrar dependiendo el entorno donde nos situemos:

[2] http://www.analytictech.com/mb874/papers/march.pdf

1. El agente se encuentra ante un entorno dinámico donde las condiciones y variables cambian constantemente, por tanto necesitaremos elegir el mejor momento en el que explorar, lo suficientemente pronto para poder adaptarnos a los cambios, y lo suficientemente tarde como para obtener un mínimo de información necesaria.

2. El agente se enfrenta a en un espacio grande de con muchos estados y acciones. En este escenario explorar cada uno de estos estados y acciones podría ser muy costoso. Por ello necesita elegir un subconjunto prometedor de estados y acciones a explorar.

En ambos casos habrá que buscar el equilibrio entre ambos factores, que nos permitirá ajustarnos a la solución del problema de una manera más eficaz.

El dilema E-E aparece en multitud de escenarios de diferente naturaleza, en algunas ocasiones con nombres distintos, para adaptar el dilema a su problema en particular. Ejemplo de ello, son:

- El dilema de *"stabilidad vs sensiblidad"* (stability vs sensitivity dilemma), presente en modelos neurocomputacionales[3] que intentan buscar el equilibrio entre el procesamiento neuronal de la información y la eficiencia del sistema.
- El problema de *"precisión vs robustez"* (accuracy vs Robustness dilemma), que podemos encontrar en modelos de mecatrónica [Sang Hoon et al., 2009], donde se busca la estabilidad entre la precisión de la impedancia y robustez frente al error de modelado.
- Los sistemas multiclasificadores, los cuales son empleados comúnmente para resolver problemas de clasificación, presentan el dilema de diversidad vs precisión (diversity vs accuracy dilemma [Diego F. et al, 2009]) donde para construir ensambladores robustos, es necesario que los clasificadores individuales sean tan precisos como diversos entre ellos.

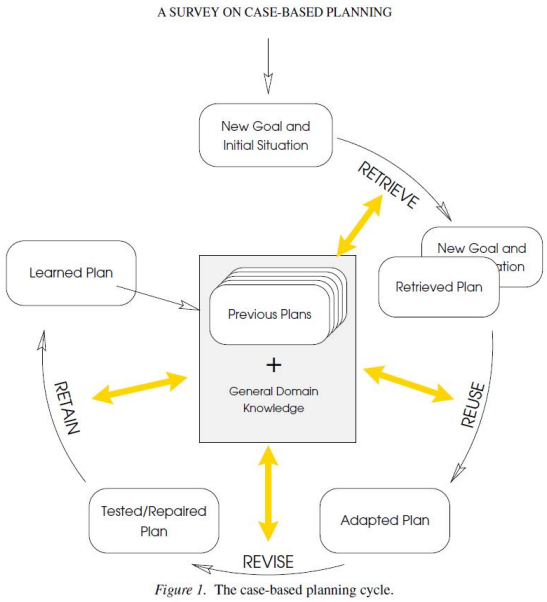


Figura 2.2: Ciclo de vida de un sistema CBR, tomada de [Spalazzi, 2001]

Otro ejemplo, que analizaremos con más detalle, es el de los sistemas de razonamiento basado en casos (CBR). Este sistema se basa en las soluciones de problemas anteriores para intentar

[3] http://inls.ucsd.edu/wlc_locust.html

solucionar nuevos problemas. En la figura 2.2 podemos ver las cuatro etapas que componen el ciclo de vida de los CBR, de las cuales nos centraremos en dos, puesto que la mayoría del trabajo en la planificación basada en casos está centrada en las fases de recuperación y reutilización, entre las que se presenta un dilema E-E. Dicha etapas se detallan a continuación:

1. Recuperación de casos (Retrieve): El objetivo de esta fase, es extraer los casos almacenados en la memoria del sistema CBR más similares al problema que se desea solucionar.

2. Reutilización de casos (Reuse): En esta etapa, se estudian las diferencias entre el caso (o casos) seleccionados en la etapa anterior y el problema presente. En esta etapa también se tienen en cuenta cuáles son las características de los casos recuperados, que pueden trasladarse al presente problema.

La reusabilidad de casos esta fuertemente determinada por la solución particular contenida en el caso y no solo en la descripción del problema. Dependiendo del dominio del problema, la recuperación de casos puede ser muy costosa cuando la memoria de la base de casos ha alcanzado un tamaño considerable. En este caso, un equilibrio entre el objetivo de encontrar el mejor de los casos y el objetivo de reducir al mínimo el tiempo de recuperación aparece, lo que hace latente su dilema E-E particular, que podemos ver gráficamente en la figura 2.3.

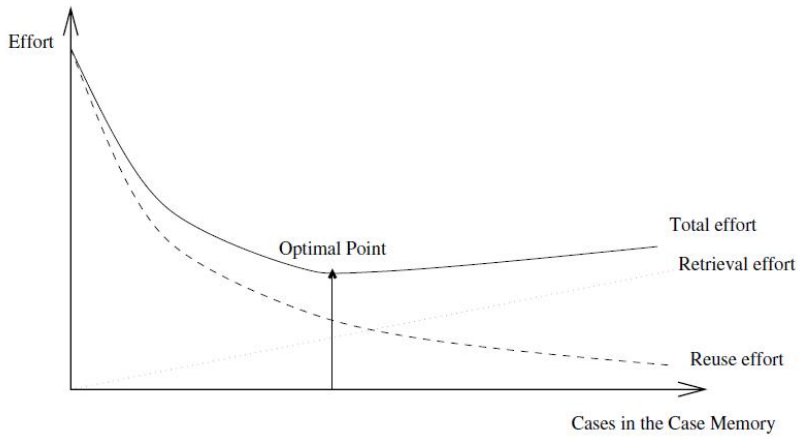


Figura 2.3: Equilibrio entre la recuperación y la reutilización de casos, tomada de [Spalazzi, 2001]

El objetivo de toda resolución en problemas en los que aparezca el dilema E-E, será el encontrar el equilibrio entre los dos posibles factores que intervengan en su resolución. En concreto en entornos multiagente tendremos irremediablemente que enfrentarnos a este dilema: aunque el entorno natural no sea dinámico, la propia dinámica social del grupo, presentará, a cada uno de los miembros del equipo, un entorno cambiante. Uno de los aspectos más interesantes es el trabajo de colaboración con intercambio de información, en la tarea de exploración de un entorno. En dichas tareas, se nos plantea el dilema de exploración-explotación. Puesto que estamos extendiendo los algoritmos de aprendizaje a sistemas multiagente, nos vamos a mover en un entorno con continuos cambios. Cada agente tendrá que ser capaz de adaptarse al comportamiento cambiante de sus compañeros, que también están aprendiendo individualmente. En esta situación, emerge una nueva forma de la paradoja E-E: es posible que un agente este constantemente explorando el entorno para integrar todos los cambios recientes a su conocimiento, sin embargo, dicha exploración no se debe hacer de manera excesiva si no queremos desplazar demasiado los resultados inmediatos. Los experimentos realizados en este proyecto están enfocados en dicho aspecto, empleando diferentes metodologías.

Capítulo 3

Marco formal y metodología

En este capítulo mostraremos un modelo propuesto para el diseño de comportamiento colectivos en entornos con incertidumbre, adaptado al entorno de trabajo UT2004. Nos inspiraremos en [Vergassola et al, 2008], donde se propone un método que máximiza el rango de ganancia de información esperada. En este modelo, denominado infotaxis, la "información" juega un papel análogo a la concentración química en la Quimiotaxis: las primeras tomas de datos no se utizan para localizar la fuente (estratregia voraz) si no para reorganizar la trayectoria de búsqueda de la manera más óptima para recavar la mayor cantidad de información, lo que a veces implica alejarse de la fuente. En la infotaxis a medida que el agente va reuniendo información, va reduciendo su estrategia de exploración y activando la de explotación. En la última parte de este capítulo se detallan las metodologías aplicadas para la implementación de nuestro modelo.

3.1. Infotaxis: inspiración y limitaciones

Se sabe que en procesos de búsqueda en la naturaleza, diferentes organismos empiezan las búsquedas de sus "datos" fuente con unos comportamientos en zigzag o mediante algún tipo de contorneo (casting) que no parecen destinados a aproximarse a la fuente, sino a muestrear el entorno. Estos comportamientos, sin embargo, se han intentado reproducir artificialmente en robots, con resultados muy pobres cuando han sido explícitamente programados. El zigzag y el casting no deben ser comportamientos programados, sino que emergen inevitablemente de cualquier estrategia general que intente maximizar la cantidad de información obtenida [Vergassola et al, 2008]. Un zigzag programado no sirve de nada: sólo es útil en circunstancias reales muy concretas, locales y transitorias. Y cuando se dan éstas, el zigzag no es un programa, sino una mera consecuencia de un algoritmo mucho más básico, profundo y general.

En la figura 3.1, podemos observar que a partir del trabajo de [Vergassola et al, 2008], el modelo infotáctico alcanza su valor su máximo para valores temporales menores, en comparación con técnicas voraces. Además los autores comparan sus resultados con las trayectorias de organismos vivos, obteniendo una alta similitud [Grasso et al, 2000] aunque esta no implique la equivalencia entre los mecanismos que generan comportamientos.

Este modelo nos va a servir como marco inspirador para plantear nuestro modelo particular, en el que consideraremos que las etapas de exploración y explotación en sistemas grupales pueden ser modeladas mediante funciones exponenciales crecientes y decrecientes, lo que simplificará el modelo infotáctico, pero consevará el fenómeno que reproduce.

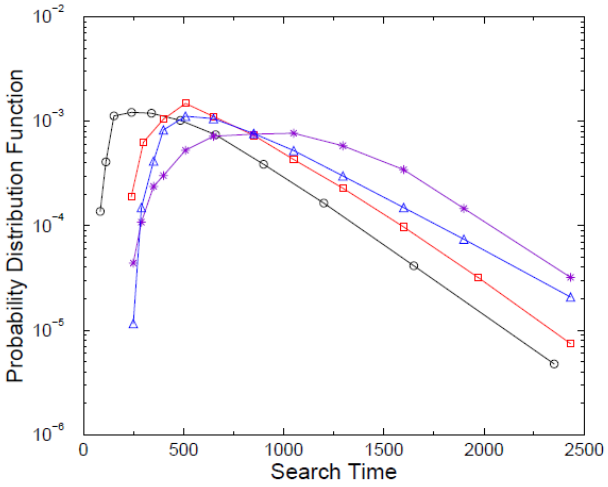


Figura 3.1: Gráfica comparativa entre funciones de distribución de probabilidad de tiempo de búsqueda para cuatro estrategias diferentes. (1) Infotaxis (negro), (2) Algoritmo voraz donde el agente elige moverse hacia estados de mayor probabilidad esperada (azul), (3) Estrategia de maximización local de la probabilidad de detección (morado), (4) Estrategia complementaria a (2) (roja). Figura tomada de [Vergassola et al, 2008]

3.2. Modelo propuesto

En el ámbito de la inteligencia artificial es habitual aceptar que, en entornos que exigen rapidez de respuesta, son necesarias soluciones menos ajustadas y más rápidas (frente a soluciones mejores pero que emplean mayor tiempo para generarse) y que la situación contraria tiene lugar cuando se dispone del tiempo necesario para generar una solución más ajustada. Esta visión asume una perspectiva sobre la inteligencia – desacoplamiento del espacio de decisión y el de acción que ha sido plenamente criticada desde hace más de una década (Spivey, 2007; Busemeyer et al, 2006), pero cuyas implicaciones no han sido del todo asumidas por falta, en muchas ocasiones, de modelos, teorías y desarrollos de carácter cuantitativo.

En nuestro sistema consideramos una función ajuste que denotamos $a(t)$, que será una medida de la capacidad media del sistema para encontrar soluciones a problemas del entorno. En otras palabras el ajuste de la solución del sistema en un instante t es una medida de similitud entre la solución puesta en práctica, $x(t)$, y la ideal $x^{\star}(t)$.

Suponemos que esta "función ajuste" en la etapa de exploración puede ser modelada mediante una exponencial creciente de la forma:

$$a(t) = a_m(1 - e^{-t/\tau})$$

que se representa según la figura 3.2.

En cambio, en la etapa de explotación, donde se emplea la información obtenida en la fase previa, el ajuste decrece según el entorno cambia. Dicha evolución se puede representar mediante una exponencia decreciente de la forma:

$$a(t) = a_m(e^{-t/\epsilon})$$

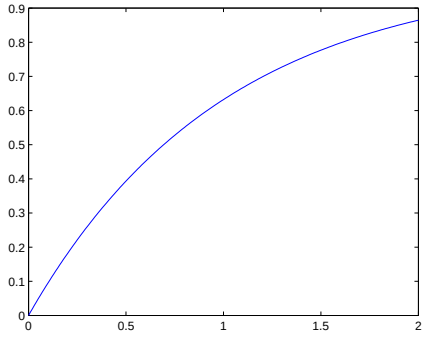


Figura 3.2: Función de ajuste en fase de exploración

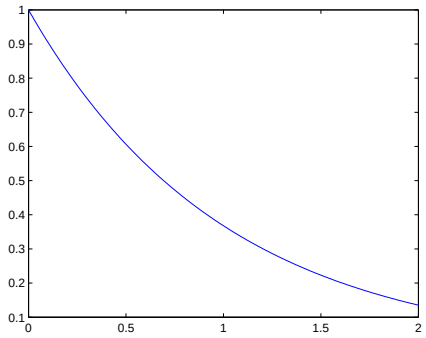


Figura 3.3: Función de ajuste en fase de explotación

representada en la figura 3.3.

La dinámica que regiría el comportamiento de un sistema multiagente cuyas etapas de exploracion-explotación sean modeladas con las expresiones arriba indicadas, sería la siguiente. Denotamos como $\gamma(t)\epsilon\{\gamma_0, \gamma_1\}$las dos fases del dilema (γ_0: exploración; γ_1: explotación).

Para el caso $\gamma(t)$=γ_0, integrando en el intervalo $t \in \{t_0, t_1\}$ tenemos:

$$\frac{d}{dt}a(t) = \frac{1}{\tau}(a_M - a(t))$$

Para el caso $\gamma(t)$=γ_1,entre $t \in \{t_1, t_2\}$ se tiene:

$$\frac{d}{dt}a(t) = -\frac{1}{\varepsilon}a(t)$$

Ambas pueden combinarse, tomando como valores $\gamma_0 = 0$, $\gamma_1 = 1$, obteniendo la ecuación global del comportamiento:

$$\frac{d}{dt}a(t) = -\frac{1}{\varepsilon}a(t) + \frac{1}{\tau}a_M(1 - \gamma(t))$$

El propósito es obtener el balance entre exploración y explotación que maximice la función de ajuste a lo largo de un periodo de tiempo $(0, T)$. El ajuste global de la solución del sistema puede expresarse mediante:

$$a(T) = \frac{1}{T} \int^T \gamma(t) \cdot a(t) \cdot dt$$

Por tanto, podemos calcular el comportamiento óptimo del sistema considerando (Figura 3.4): (a) una estrategia voraz , (b) una estrategia tipo infotaxis.

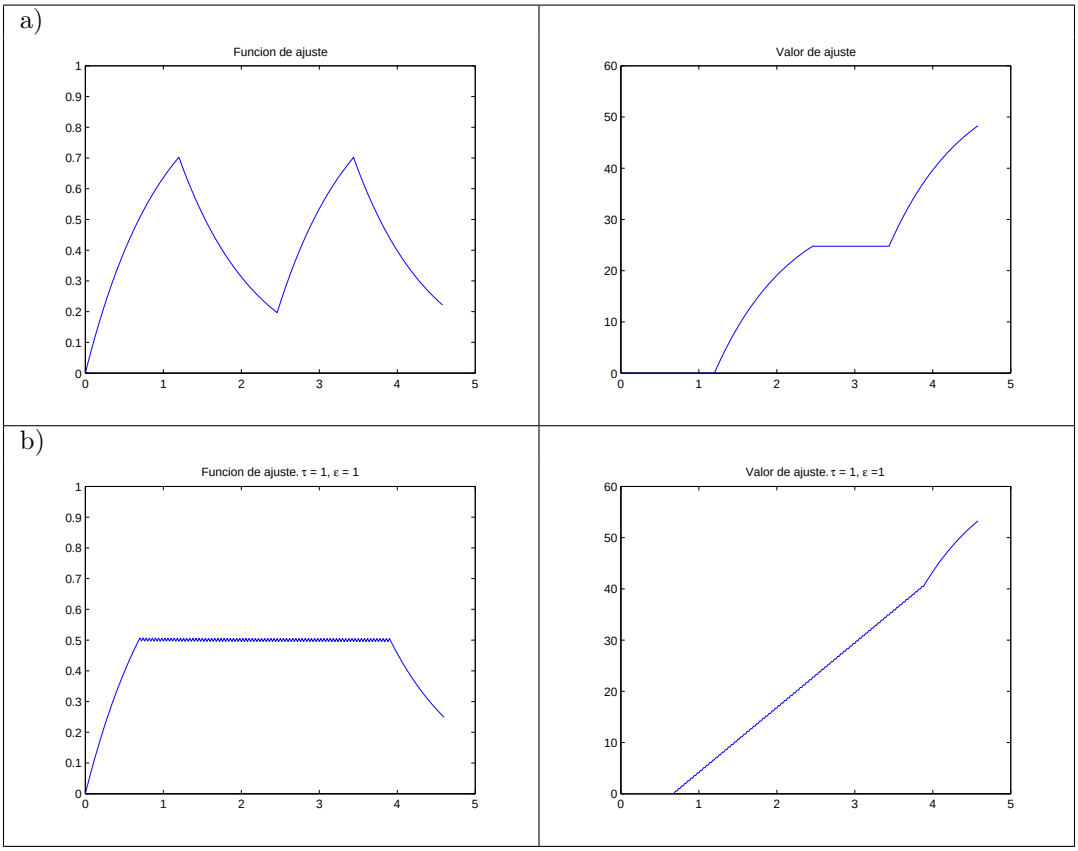


Figura 3.4: Comparación de estrategias. Representación de la función de ajuste $a(t)$, y el ajuste global $a(T)$ para (a)la estrategia voraz, y (b) el modelo tipo infotaxis. Podemos comprobar que el ajuste global es mayor para el segundo modelo.

Para diferentes valores de (τ, ε) obtenemos cualitativamente el mismo fenómeno para diferente niveles de ajuste global. En la figura 3.5 se representan tres casos para diferentes valores.

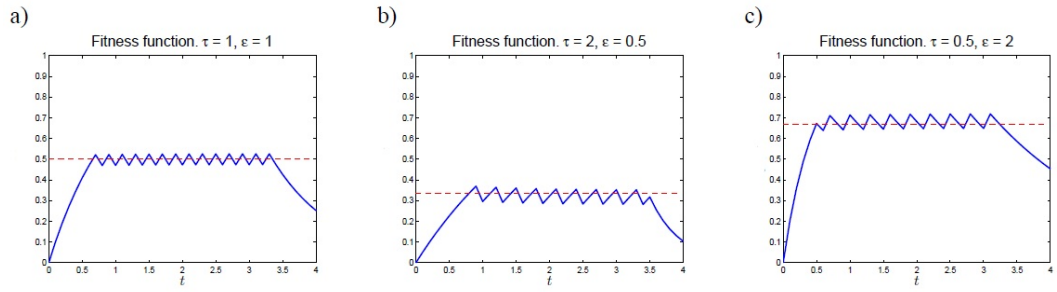


Figura 3.5: Representación del ajuste global $a(t)$ para diferentes parejas de valores τ, ε

Si representamos la distribución de valores de ajuste global en términos del ratio $\varepsilon(t)/\tau(t)$, obtenemos lo mostrado en la figura .

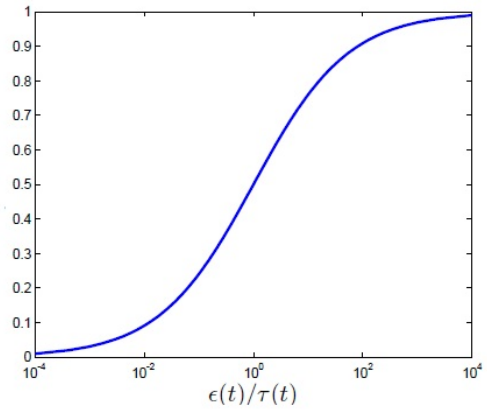


Figura 3.6: Distribución de valores de ajuste global $a(T)$ en términos de los parámetros ε y τ

En los videojuegos actuales, las técnicas de simulación de comportamiento grupal no son modeladas de manera realista [von Mammen et al, 2009]. Nuestro interés a continuación es poder representar, según el modelo propuesto, el comportamiento de sistemas colectivos en un entorno de simulacion de videojuegos (UT2004), que permita darle a los agentes una apariencia más "humana". Para ello, emplearemos diferentes metodologías en la fase de implementación. En la siguiente sección se describen brevemente.

3.3. Metodologías aplicadas

3.3.1. Swarming

El comportamiento de algunos seres vivos, ha sido durante años objeto de estudio a la hora de programar algoritmos para resolver problemas. El "swarming" es un ejemplo de estos algoritmos, el cual se puede adaptar a diferentes sistemas, optimizando o resolviendo problemas de diferente naturaleza.

El swarming se observa frecuentemente en comportamientos colectivos de estudios biológicos en insectos, donde éstos resuelven problemas complejos trabajando en equipo. Las hormigas son unos de los insectos a los que se les asocia este comportamiento, que emplean para resolver la problemática de trazado de caminos a los hormigueros y a las fuentes de comida. El movimiento de las hormigas es guiado por el flujo continuo de estos insectos, que van dejando un rastro de feromona a su paso. Así, las demás hormigas siguen ese rastro para guiarse, el cual se hace cada vez más consistente por el paso de más hormigas. A su vez, en las zonas por las que ya no pasan más hormigas, el rastro de feromona se va diluyendo a una cierta velocidad, dependiendo del entorno, hasta que desaparece.

El comportamiento swarm fue simulado por primera vez en un computador en 1986, mediante el programa Boids[1]. Este programa simulaba a un conjunto de agentes simples que eran capaces de moverse acordes a una serie de reglas básicas.

[1] http://www.red3d.com/cwr/boids/

3.3.2. Algoritmos genéticos

Aunque las primeras nociones de algoritmo genéticos pueden encontrarse hace más tiempo, es reconocido que fueron impulsados definitivamente por los trabajos de John Holland [Holland, 1992]. Se consideran una técnica destacada en el contexto de la computación evolutiva. Son mecanismos de optimización o búsqueda en un espacio de estados multidimensional, cuya mayor utilidad se demuestra en problemas en los que la solución es el resultado de un proceso no lineal, aplicado a un conjunto de parámetros que definen dicho espacio de estados multidimensional [Goldberg, 1989]. Dicho conjunto de parámetros definen una posible solución al problema, y son un punto en el espacio de estados, denominado también genotipo. En este tipo de problemas es imposible aislar las variables para obtener su valor óptimo por separado.

El proceso general de evolución simulada parte de una población de individuos que codifican en su material genético (genotipo) un comportamiento o morfología como una posible solución a un determinado problema que se quiere optimizar. Esta población de individuos no son mas que un conjunto de posibles soluciones al problema (puntos en el espacio de estados). Los individuos se someten a lo largo de varias generaciones a procesos de selección, para procrear y pasar su material genético a siguientes generaciones. Tal selección requiere asignar un valor numérico a cada individuo de la población, que será una medida de lo que se acerca la solución óptima. Este valor se denomina "valor de adecuación" o "fitness", y se calcula con una función del mismo nombre. La elección de la función fitness tiene consecuencias muy importantes en la posibilidad de evolución del bot, dinámicas del proceso evolutivo y por último en la salida del proceso evolutivo. Desafortunadamente, no hay una manera de definir una función de adecuación a partir de una descripción del resultado esperado. Normalmente, (1) uno define una función basada en su propia experiencia, (2) a continuación, prueba su idoneidad a base de prueba y error (cuyo consumo de tiempo es uno de los mayores problemas en la evolución de agentes), y (3) Modifica gradualmente su valor introduciendo variables y constantes adicionales. Estas variables y constantes no son fáciles de elegir, ya que no hay un conocimiento total del comportamiento del robot una vez ha evolucionado. Podría decirse que diseñar una función de adecuación aplicable para un comportamiento deseado es normalmente más fácil que diseñar el programa correspondiente. No obstante el grado de conocimiento de un comportamiento esperado es inversamente proporcional a la necesidad de aplicación de estrategias evolutivas. Si el espacio de estados es de dimensión n, el "paisaje de adecuación" (fitness landscape) es un conjunto de puntos en un espacio de $n+1$ dimensiones, que puede ser continuo y formar una superficie (como un paisaje), o ser rugoso o completamente inconexo, lo que influye en la capacidad de búsqueda en el algoritmo genético. De entre todas las soluciones posibles se escogen, en función de su adecuación, las mejores y se genera la descendencia, que sustituye a los individuos no elegidos o a todos los de la población anterior. Se puede mantener una elite de individuos que pasa a la siguiente generación sin modificaciones.

Al igual que en la evolución natural, a lo largo de las generaciones irán apareciendo individuos u organismos mejor adaptados en su comportamiento a su entorno virtual, con una morfología más adaptada al entorno en el que actúa y las tareas que debe realizar, o sencillamente, más cercanos a una solución buena al problema a optimizar.

En la figura 2.1 podemos ver los pasos típicos que se realizan en un algoritmo genético, cuyas fases se detallan a continuación:

- Población inicial: La población inicial de individuos se genera de forma aleatoria, considerando los valores mínimos y máximos de cada variable.
- Evaluación: Para cada individuo de la población, se simula en el entorno correspondiente y obtenemos el valor de adecuación para cada agente.
- Selección: Este operador elige aquellos individuos que van a generar nuevos descendientes, teniendo en cuenta el valor fitness, de modo que serán seleccionado con mayor probabilidad aquellos con un mejor valor. Existen varios métodos para realizar la selección, nosotros

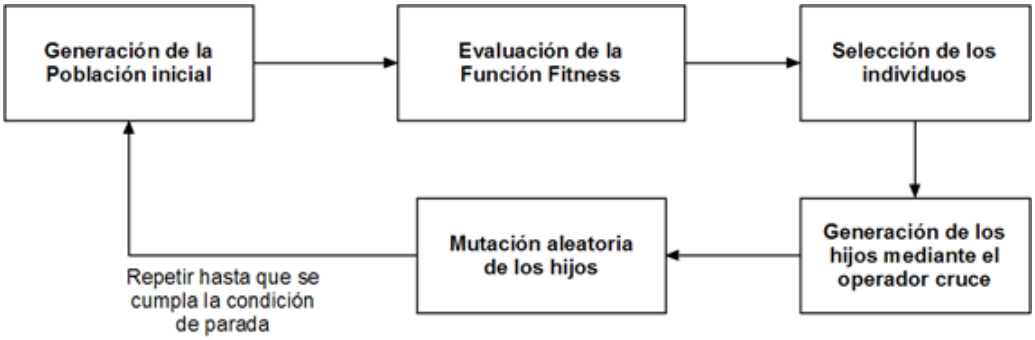


Figura 3.7: Etapas típicas de un algoritmo genético

emplearemos la "selección por torneo". Este método consiste en determinar subconjuntos de individuos, elegidos de modo aleatorio entre la población, y se elige para procrear el mejor de ese subconjunto.

- Cruce (Crossover): En esta fase, se combina el operador genético de dos progenitores previamente seleccionados por el operador anterior. El método aplicado en este operador dependerá de la codificación del genotipo de cada individuo.
- Mutación: Este operador cambia el contenido del material genético de un determinado cromosoma. En el caso de una codificación del genotipo de tipo binario, una mutación podría ser el cambio de un bit.

Algoritmos genéticos en sistemas multiagente: Clonal vs Aclonal

En los últimos años un buen número de investigadores han aplicado satisfactoriamente la evolución artificial en bots autónomos (e.g. Baldassarre et al., 2003a; Botee and Bonabeau, 1998; Luke and Spector, 1996; Mitchell et al., 1996; Wu et al., 1999). Sin embargo, se han centrado casi exclusivamente en sistemas individuales. El paso de sistemas individuales a sistemas multiagente lleva asociado consigo una serie de problemas metodológicos. Tal vez el más básico de estos problemas surge al plantearnos la siguiente cuestión: ¿Como deberían organizarse las generaciones y pruebas de los distintos agentes? La respuesta a esta pregunta nos introduce en una comparación entre las dos posibles aproximaciones: clonal y aclonal.

La implementación clonal, es una simple variación de la evolución genética que se aplica a sistemas individuales. Los genotipos son evolucionados en una única población, con cada genotipo codificando los parámetros necesarios para definir el comportamiento de cada uno de los agentes del sistema. La evaluación de un genotipo se realiza de la siguiente manera: Primero, el genotipo es codificado para producir un elemento del sistema. Después ese elemento es clonado hasta que el sistema tiene el tamaño que queremos. Por útlimo, todo el sistema es evaluado acorde con un criterio apropiado, y se la asigna un valor fitness al genotipo.

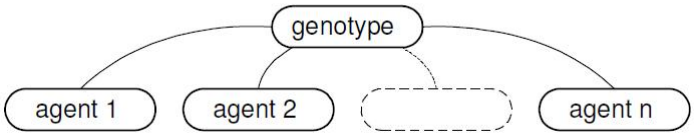


Figura 3.8: Aproximación clonal

Con esta estrategia obtenemos un sistema que está especificado por el material genético de un solo individuo; por tanto, desde la perspectiva de evolución, un sistema clonal es el fenotipo de un sólo genotipo. Es decir, un solo individuo distribuido.

El método clonal ha sido empleado por muchos investigadores; entre los que podemos destacar a Bottee y Bonabeau (1998), los cuales usaron esta aproximación para evolucionar propiedades de una colonia de hormigas, para resolver el problema del vendedor viajero. Las algoritmos de colonias de hormigas son paralelos y distribuidos, con agentes idénticos, inspirándose en los rastros de feromona de las hormigas reales.

Por otro lado tenemos la aproximación aclonal. Dos características principales definen esta implementación: En primer lugar, al igual que en la aproximación clonal, tenemos una única población, donde cada genotipo en la población codifica el comportamiento de un agente. La siguiente característica, es que cada agente esta especificado por un genotipo distinto; por tanto, cada agente necesita un genotipo para ser codificado.

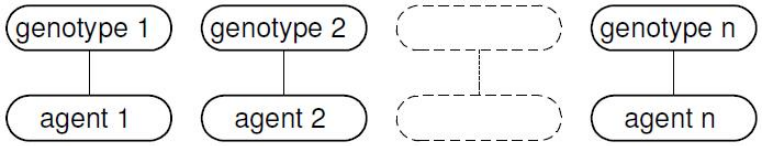


Figura 3.9: Aproximación aclonal

Con la implementación aclonal, se esta favoreciendo la creación de roles o subgrupos dentro de los individuos.

Este sistema se ha empleado con mucha frecuencia para investigar la evolución de comunicaciones naturales entre varios agentes (Bullock,1998, Quinn an Noble,2001).

En la sección siguiente propondremos dos tipos de experimentos: (1) con técnicas clonal y (2) técnicas aclonal, con el objetivo de comprobar la versatilidad de cada una de estas metodologías.

Capítulo 4

Herramientas utilizadas

4.1. Unreal Tournament 2004

Unreal Tournament 2004 (UT2004) es un videojuego de acción tipo FPS (first-person shooter), como podemos ver en la figura 4.1, desarrollado por Epic Games and Digital Extremes. Está principalmente orientado a la experiencia multijugador, aunque también existe el modo de un solo jugador, el cual emula el juego multijugador a través del uso de bots controlados por la computadora. Pero tal como UT2004 anuncia en su pantalla de inicio: "GAME EXPERIENCE MAY CHANGE DURING ONLINE PLAY"; Si queremos disfrutar del juego plenamente, deberemos jugar en modo multijugador a través de la red, puesto que la experiencia de juego dada por los bots controlados por el juego, no tiene nada que ver con jugar contra jugadores humanos. Cada vez con más interés se pretende que la IA aplicada a videojuegos permita mostrar bots con un comportamiento humano y por tanto realista, lo que ha día de hoy todavía no se ha conseguido completamente en ningún videojuego.

Existen varios modos de juego en UT2004, los cuales tienen objetivos diferentes y diferente mapas. Los formatos existentes son los siguientes:

- Capture the Flag (Captura la Bandera). Modo de juego por equipos en el que cada grupo tiene una base y una bandera que defender. El objetivo es robar la bandera del enemigo y llevarla hasta la base de equipo contrario.
- Deathmatch (Todos contra todos). Modo de juego individual en el que gana el jugador que más muertes registre en un tiempo determinado o que alcance un nivel predefinido de muertes de jugadores rivales.
- Team Deathmatch (Combate por equipos). En este modo, las reglas son las mismas que en la modalidad Deathmatch, pero en este caso el marcador será la suma de los resultados individuales de todos los miembros del equipo.
- Double Domination (Doble dominación). Similar al modo Capture the Flag, este modo por equipos consiste en controlar de manera presencial y simultánea, la base del propio equipo y del contrario durante un tiempo determinado.
- Bombing Run (Carrera de bombardeo). Modalidad por equipos con un esquema similar al de un partido de fútbol adaptado a los escenarios de UT04. Gana el equipo que consiga pasar una bola por un portal un número determinado de veces en un tiempo prefijado.

- Last Man Standing (Último hombre en pie). Similar al modo Deathmatch, pero en este caso cada jugador participa con un número limitado de vidas en la partida.

Aunque por la temática de este proyecto, parezca que los modos de juegos por equipos son los más indicados, en realidad lo qué más se ajusta al objetivo de nuestros experimentos es el modo "Capturar la Bandera". En este modo de juego, también existe un equipo y además, puesto que el objetivo de nuestros experimentos girará alrededor de la exploración del entorno, podremos utilizar las banderas como puntos clave.

Figura 4.1: Unreal Tournament 2004

En la actualidad existen muchos videojuegos donde el juego en equipo está contemplado. Similares a Unreal tournament podemos encontrar el popular videojuego Counter strike y también el no menos conocido, Quake. Hemos elegido Unreal tournament por varios motivos, que se enumeran a continuación:

- Mantiene su código abierto a todos los usuarios, lo que permite codificar nuestras propias implementaciones de bots a traves de su lenguaje script, Unreal Script, así como la creación de nuevos mapas con la herramienta UnrealEd, incluida en el juego.

- Existe la herramienta Pogamut, desarrollada por investigadores ajenos a la empresa del videojuego, pero que nos simplificará la tarea de crear un bot. Esta herramienta se explicará con detalle en la próxima sección.

- Es la herramienta empleada en el concurso a nivel mundial 2kBotPrize, celebrado todos los años, donde investigadores de todo el mundo presentan sus bots con el objetivo de superar una adaptación al test de Turing.

Todos estos puntos parecen indicar que estamos ante la herramienta adecuada donde diseñar nuestros experimentos de inteligencia artificial.

4.2. Pogamut

Pogamut es una plataforma de código abierto usada para el rápido desarrollo de comportamientos en agentes virtuales incrustados en un entorno 3D del videojuego Unreal Tournament 2004. Esta herramienta se integra con el entorno de programación NetBeans como un plugin adicional y permite definir el comportamiento de los agentes mediante el uso de Java.

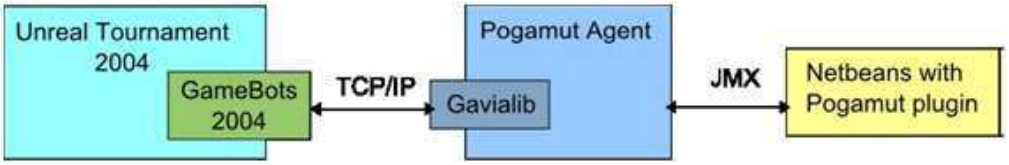


Figura 4.2: Arquitectura de Pogamut

Pogamut es el encargado de las siguientes tareas:

- Traducir la información procedente del servidor mediante el uso de un traductor (o parser) local. El módulo traductor es un elemento que precede al cliente y permite simplificar el procesamiento de los mensajes de Gamebots, convirtiéndolos de su formato ASCII original, a objetos propios del lenguaje Java. Además, el traductor se encarga de reducir el flujo de información que llega al cliente, puesto que sólo le envía aquella que ha sufrido modificaciones desde el último envío.
- Devolver al cliente toda la información del agente relativa a su propio estado o a la situación del entorno que lo rodea en cada momento de la partida. Esta gestión de información, queda reflejada en NetBeans, ya que dicha herramienta se encarga de mostrar los registros del agente obtenidos por Pogamut. La información que llega a dichos registros del agente, puede ser síncrona (toda aquella procedente de llamadas realizadas por el usuario) o asíncrona (la notificación de eventos ocurridos a lo largo de la partida en momentos puntuales en imprevisibles y que pueden requerir de un trato especial por parte del agente, si así está definido en su comportamiento).

Gamebots es el módulo encargado de la gestión de información en el lado del servidor. Gamebots es una extensión de UT2004 que permite controlar los distintos agentes autónomos virtuales a través de Unreal Script, mediante una interfaz de comunicación TCP/IP que utiliza mensajes de texto en formato ASCII para transmitir datos. De este modo, no sólo se pueden controlar los agentes del juego mediante comandos de texto, sino que se recibe información relativa al estado de la partida en cada instante de tiempo.

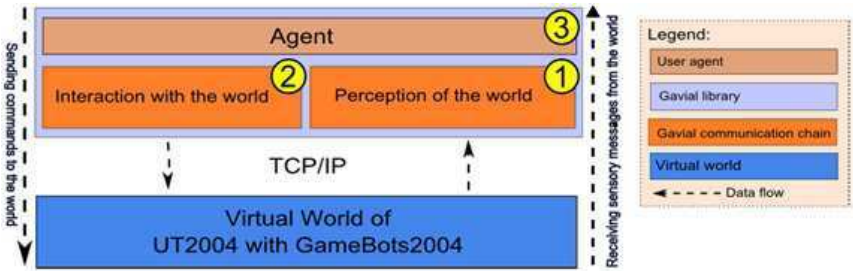


Figura 4.3: Arquitectura de GaviaLib

GaviaLib es una librería dentro de Pogamut que se encarga de traducir UnrealScript y pasarlo a Java. Permite conectar agentes a casi cualquier entorno virtual. En la Figura podemos ver su arquitectura a alto nivel. (1) Procesa los mensajes del entorno. (2) Envía los comandos al mundo virtual. (3) Interfaz del agente.

4.3. UnrealED

Una de las características centrales y definitivas de Unreal Tournament 2004, y uno de los mayores componentes de su éxito, es la facilidad con la cual muchos jugadores pueden crear y compartir sus propias modificaciones y contenido propio tales como mapas. Varios de los mapas creados por jugadores suelen ser superiores a aquellos que vienen con el juego. En consecuencia, muchos, sino la mayoría, de los servidores de UT alrededor del mundo incluyen mapas de terceros en sus rotaciones. Los mapas personalizados pueden ser creados con el editor UnrealED incluido con el juego.

Nosotros emplearemos la herramienta para construir un mapa sencillo que se adapte a las características de nuestros experimentos y nos permita ver de forma clara los resultados.

Figura 4.4: Mapa creado en UnrealED

4.4. Diseño del bot en el entorno UT2004

El diseño del bot, es decir, su capacidad de percibir su entorno y de ejecutar sus acciones, viene limitado por las posibilidades del entorno de simulación utilizado. A la hora de abordar el diseño de nuestro agente, deberemos tener en cuenta lo que Unreal Tournament 2004 y Pogamut nos permiten hacer. Tendremos que conocer de que manera UT2004 nos posibilita percibir el entorno, así como las posibles acciones que nuestro bot puede ejecutar.

Podemos asemejar el comportamiento del bot con el modelo clásico de representación de la mente, el cual podemos ver en la figura 4.5. La capa de razonamiento quedará de lado del programador, dónde usando las entradas (percepción) facilitadas por el entorno implementaremos un comportamiento para nuestro bot, generando una salida a través de las acciones posibles.

Figura 4.5: Modelo de representación clásico de la mente

Percepción

Existen dos modos básicos de percepción para un bot, en el entorno Unreal: percepción por "Raycasting" o por mapas de navegación.

El RAYCASTING es un método de percibir el entorno que se basa en rayos, a través de los cuales veremos sus intersecciones con el mapa. Podemos configurar el número de rayos que queramos que nuestro bot lance, así como su longitud. En cada rayo y en cada momento, el bot nos devolverá un valor binario que nos notificará si el rayo correspondiente está detectando algún objeto, ya sea un bot enemigo, un bot amigo o una pared.

La percepción a través de MAPAS DE NAVEGACIÓN se apoya en el hecho de que los mapas en UT2004 están cubiertos por nodos llamados puntos de navegación (NavPoints). El bot es capaz de saber en todo momento en qué punto se encuentra, así como dónde se encuentran los demás puntos en el mapa. En mapas de Unreal bien construidos, cada navpoint está situado en un sitio seguro y alcanzable por el bot. Los puntos conectados están unidos por una línea. Existen más elementos que desarrollan la percepción del agente, los cuáles no vamos a emplear en nuestros experimentos, pero se pueden consultar en el anexo A: Manual de Pogamut.

Razonamiento

Esta es la capa que corresponde a la implementación del bot, dónde usaremos los datos proporcionados por la capa perceptiva para modelar el comportamiento de nuestros agentes inteligentes a través de las acciones que éstos pueden realizar. Para manejar los resultados obtenidos por un bot perceptivo de tipo Raycasting, podemos hacerlos de dos maneras: Creando un listener (Ver glosario), el cual nos notifique cuando un rayo ha recibido alguna señal del entorno o bien comprobando la presencia de algún objeto de forma periódica. Para usar Raycasting en nuestro bot, debemos seguir dos pasos:

1. Activar raycasting y visualización de rayos: podemos hacerlo de dos formas, o bien computar continuamente las intersecciones de los rayos con el mundo (Autotrace=true) o bien denir nuestros rayos y calcular solo las intersecciones que nos interesen (DrawTraceLine= true).
2. Inicializar rayos: definimos los rayos que deseemos a través de AddRay, pasandole como parámetros principales el nombre, el vector de dirección y la longitud del rayo. Podemos crear tantos rayos como queramos.

En el caso de un bot que base su percepción en mapas de navegación, recibiremos a través de un listener el estado del mismo. Para que el bot sea capaz de ir de un sitio A a un sitio B, vamos a necesitar una planificador de ruta: Path planner. En Pogamut existen distintos interfaces de Path planner, siendo UT2004A StarPathPlanner el planificador por defecto. Este planificador usa el algoritmo A* para el calculo de rutas, siendo exactamente el mismo que el de los bots nativos de UT. Tambien existe otra implementación declarada en FloydWarshallPathPlanner, la cual precalcula todas las rutas posibles entre todos los nodos al principio, lo cual tiene un coste inicial considerable. Cabe decir que podemos implementar nuestro propio algoritmo planificador. Una vez tenemos la ruta calculada ya solo nos queda ejecutarla, de ello se encarga el Path executor. Este módulo contiene el Path navigator que es el que se encargar de recorrer la ruta calculada anteriormente, evitar obstáculos, abrir puertas, esperar ascensores, etc. El navegador por defecto es UT2004PathNavigator. En este caso, también podemos crear nuestro propio navegador.

Acciones

Las acciones son operadores que pueden llamarse desde la capa de implementación para provocar cambios en el estado del mundo en el que nos encontramos. El movimiento será la acción más empleada en nuestros experimentos, debido a la naturaleza de los mismos, pero Pogamut nos permite llevar a cabo un gran número de acciones que podemos ver en la tabla 4.2.

Act	AddBot	AddInventory
AddRay	Combo	CommandPlayer
Configuration	ConfigurationObserver	Console
ContinuousMove	DialogBegin	DialogCancel
DialogEnd	DialogItem	DisconnectObserver
Dodge	DriveTo	EndPlayers
EnterVehicle	FactoryUse	FastTrace
GetAllInvetories	GetAllNavPoints	GetAllStatus
GetGameInfo	GetItemCategory	GetMaps
GetPath	GetPlayers	GetSelf
GetSpecialObjects	GetVisibleObjects	GiveInventory
ChangeAttribute	ChangeMap	ChangeTeam
ChangeWeapon	CheckReachability	Initialize
InitializeObserver	Jump	Kick
LeaveVehicle	Move	PasswordReply
Pause	Pick	Ping
PlaySound	Quit	Ready
Record	RemoveRay	Respawn
Rotate	SendMessage	SetCrouch
SetDialog	SetGameSpeed	SetLock
SetPassword	SetPlayerControl	SetRoute
SetSendKeys	SetSkin	SetWalk
Shoot	ShowText	SpawnActor
StartAnimation	StartPlayers	Stop
StopRecord	StopShooting	Throw
Trace	TurnTo	

Tabla 4.2: Acciones posibles del bot en Pogamut

Capítulo 5

Experimento 1 - siSosiG Bot

Should I stay or should I go now?
If I go there will be trouble
And if I stay it will be double
So come on and let me know

(The Clash)

5.1. Descripción del problema

En este experimento veremos como un equipo de bots en el entorno UT2004, modelado mediante la propuesta presentada en el capitulo 3, es capaz de resolver el problema de encontrar la bandera utilizando técnicas de comunicación a larga distancia. Emplearemos un algoritmo genético para encontrar que combinaciones de bots creen un sistema más eficiente.

El experimento se desarrolla en un escenario de encontrar la bandera, en el que el objetivo final es que todos los bots lleguen a la bandera enemiga en el menor tiempo posible, sin conocer previamente ningún dato sobre el mapa. El objetivo de cada bot del equipo es, en primer lugar, encontrar la bandera y posteriormente avisar al resto de sus compañeros para que acudan a ella, como podemos ver en la figura 5.1. Así pues, cada bot debe escuchar las transmisiones de sus compañeros mientras sigue con la búsqueda individual de la bandera, no pudiendo realizar ambas tareas a la vez. Por este motivo, nos encontramos con el dilema de exploración-explotación planteado en capítulos previos.

Nuestros agentes se moverán por todo el mapa de manera aleatoria en busca de la bandera, pero también podrán esperar a que otros bots la encuentren. Por tanto cada bot tiene dos estados:

- Escuchando: Cuando se encuentra en este modo, el bot es capaz de recibir transmisiones de sus compañeros y se encuentra parado. Por lo tanto, el bot una vez avisado por un compañero de la localización de la bandera, acudirá inmediatamente.
- Rastreando: En este estado el bot se encuentra en movimiento, buscando por puntos aleatorios la bandera. El agente puede encontrar la bandera y avisar a sus compañeros, pero si el es avisado por otro bot, la transmisión no la recibe hasta que cambia al estado *escuchando.*

Figura 5.1: Representación del algorimto siSosiG

Estos dos estados representan de forma particular los dos factores dados en el dilema E-E. Tendremos que buscar el equilibrio de ambos para obtener un sistema óptimo en la resolución de la tarea.

5.2. Implementación

Para nuestro experimento hemos creado bots perceptivos con un sistema de mapas de navegación, por lo que cada bot será consciente del punto en el que se encuentra o los destinos posibles a los que puede moverse. De esta manera tendremos más control sobre el movimiento de nuestro bot y no generaremos demasiadas rutas, lo que implicaría aumentar considerablemente los tiempos de simulación. Cada agente podrá realizar movimientos a los puntos de navegación adyacentes al nodo en el que se encuentra.

Implementaremos dos funciones principales para ayudar en el movimiento del bot:

- NavPoint calcularDestino (Collection<NavPoint> vecinosNavPoints, NavPoint anterior): Esta función es llamada cada vez que un bot esta en el estado 'rastreando' y se encuenta en un NavPoint. Le pasamos como parámetro la lista de todos los puntos de navegación vecinos al que nos encontramos y el NavPoint en el que hemos estado en el movimiento anterior para evitar volver al mismo. La función nos devolverá un punto aleatorio de la lista de vecinos, al que el bot se dirigirá
- NavPoint irABandera(): Esta función se llamará cuando un bot se encuentre en estado 'escuchando' y alguno de los otro agentes encuentre la bandera. La función devolverá el punto de navegación donde se encuentre la bandera y el bot acudirá a ella.

Para que el bot sea capaz de ir del punto de navegación en el que se encuentra al nodo destino, vamos a necesitar una planificador de ruta. Como hemos explicado en el capítulo anterior, Pogamut nos ofrece Path planner. Este planificador usa el algoritmo A* para el calculo de rutas. Una vez tenemos la ruta calculada solo nos queda ejecutarla, de ello se encarga el Path executor. Este módulo contiene el Path navigator que es el que se encargar de recorrer la ruta calculada anteriormente, evitar obstáculos, abrir puertas, esperar ascensores, etc. Una vez que se ha ejecutado el Path executor, podemos esperar que sucedan tres cosas, las cuales capturaremos con un listener (Ver Anexo: Manual de Pogamut):

- TARGET_REACHED: Este es el caso que sucede la mayoría de las veces. El bot ha llegado correctamente a su destino, de modo que si una vez llegado a este punto su estado aún es 'rastreando', le ordenaremos que calcule el siguiente punto a moverse a través de la función calcularDestino();
- STUCK: Cuando el listener nos devuelve este caso, nos indica que nuestro bot se ha atascado, lo cuál puede ser debido a una mala construcción del mapa. Lo que haremos será buscar un nuevo destino.
- PATH_COMPUTATION_FAILED: No se ha podido ejecutar la ruta. En este caso informaremos del error y detendremos la ejecución.

Para implementar los dos posibles estados del bot (escuchar y rastrear) hemos definido un parámetro que controla el estado del mismo, el cual hemos denominado *índice de hiperactividad (IH),* puesto que define el porcentaje de tiempo que un bot esta parado o en movimiento. Los valores de dicho índice oscilan entre 0 y 1, aplicándose sobre un tiempo de ciclo (T_c), lo que nos da intervalos de tiempos en ambos estados:

- Tiempo rastreando: $IH \cdot T_c$
- Tiempo escuchando: $(1 - IH) \cdot T_c$

Cada bot podrá moverse libremente por el mapa, de manera totalmente aleatoria, cuando su estado sea 'en movimiento'. En el estado 'escuchando' el bot estará parado esperando a que algún compañero suyo le comunique que ha encontrado la bandera. Si no es así, volverá a empezar un nuevo ciclo.

Para implementar la funcionalidad de los dos posibles estados, emplearemos la función logic() que nos ofrece Pogamut. Es equivalente a la función Timer de Java, puesto que se ejecuta periódicamente, teniendo que especificar el tiempo de cada periodo. Cada vez que se ejecute la función comprobaremos el estado en el que se encuentra el bot y según su IH y el T_c, con las fórmulas definidas anteriormente, mantendremos o cambiaremos el estado del bot según corresponda.

En nuestro experimento hemos creado un escenario con una malla 25 NavPoints, un equipo con 3 bots y un $T_c$de 10 segundos.

5.3. Estudio analítico

Como podemos deducir del apartado anterior, existirán dos factores que marquen el resultado de cada simulación:

1. La probabilidad de que un bot encuentre la bandera por su cuenta.
2. La estructura del equipo de bots, teniendo en cuenta el valor IH de cada uno de ellos, que afectara al tiempo en que los bots restantes alcancen el objetivo.

La probabilidad de que un bot encuentre la bandera, la cual vendrá definida por una función del tipo:

$$p(bandera) = p(x_0, N, IH)$$

donde x_0 representa el punto donde se encuentra el bot, N el número total de puntos de navegación en la malla y IH es el índice de hiperactividad del bot en cuestión. Como nuestro mapa posee 25 puntos de navegación y 3 puntos de salida para los 3 bots que forman el equipo, la única variable en la función sería el IH del bot. Por tanto habría que calcular la probabilidad que tiene de encontrar la bandera cada bot del equipo según su IH, y luego combinar las distintas probabilidades para obtener la probabilidad global.

5.3.1. Probabilidad de encontrar la bandera

Los movimientos de nuestros bots se realizan a puntos adyacentes, con la probabilidad de moverse a cualquiera de esos puntos $1/n$, siendo n el número de puntos de navegación a los que nos podemos mover con distancia 1. Una vez que nos hemos movido aleatoriamente a uno de esos puntos, no tiene relevancia de dónde hemos venido, solo importa dónde estamos actualmente. Por este comportamiento podríamos asociar al proceso probabilístico del movimiento de nuestro bot a una cadena de Markov.

En estadística, se conoce como cadena de Markov[1] a un tipo especial de proceso estocástico discreto en el que la probabilidad de que ocurra un evento depende del evento inmediatamente anterior. Tal suceso se puede representar matemáticamente de la siguiente manera:

$$P(s_{t+1}, t+1 \mid s_t, t), \quad \forall t \in \mathbb{N}$$

En primer lugar vamos a ver como está construida nuestra malla de puntos de navegación. En la figura 5.2 podemos ver una malla 25x25 que corresponde al mapa creado para las simulaciones de este experimento. Los puntos de salida para los bots están marcados en verde (20, 21, 22) y el objetivo en rojo (7).

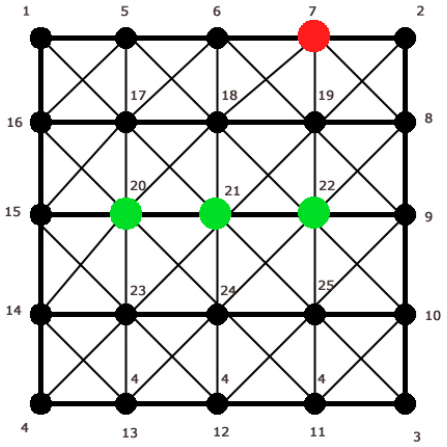


Figura 5.2: Malla de puntos de navegación para el mapa de pruebas del experimento siSosiG en UT2004

Siguiendo la definición matemática de Markov, vamos a crear una matriz 25x25, representando la probabilidad de ir un punto a otro. Las filas de la matriz representan el punto donde nos encontramos actualmente, y las columnas al punto donde nos queremos mover. Viendo la construcción del mapa en la figura, podemos distinguir tres tipos de puntos de navegación que tendrán distintas probabilidades de moverse a sus NavPoints adyacentes:

1. Nodos esquina (1-4): Probabilidad 1/3.
2. Nodos laterales (5-15): Probabilidad 1/5.
3. Nodos centrales (17-25) Probabilidad 1/8.

Construiremos la matriz donde se rellenarán las probabilidades de ir de un punto a sus adyacentes según los datos de probabilidad anteriores. Los puntos que no se encuentren alcanzables a distancia 1, serán nulos:

[1] http://es.wikipedia.org/wiki/Cadena_de_Markov

$$M=\begin{pmatrix} 0 & p_{1,2} & \dots & p_{1,n} \\ p_{2,1} & 0 & \dots & p_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \dots & 0 \end{pmatrix} \text{donde } p_{x,y} = 0 \iff distancia(x,y) \neq 1$$

Una vez que tenemos la matriz, la probabilidad que representará el éxito en alcanzar la bandera según el número de pasos que el bot realiza, vendrá expresada por:

$$M^n = M \cdot M \cdot \ldots \cdot M, \text{ siendo } n \text{ el número de pasos.}$$

En nuestro caso aproximamos $(IH \cdot T_c) \cdot n$ a la unidad, puesto que el tiempo de navegación entre nodos es aproximadamente un segundo, lo que corresponde con un movimiento, donde IH es el índice de hiperactividad del bot, T_c el tiempo de ciclo y n el número de ciclos.

Queremos llegar al punto de navegación donde se encuentra la bandera (punto 7, ver figura 5.2), pero si dejamos la matriz tal cual está e calculamos la probabilidad de ir de un punto A al punto destino en n pasos, estaremos viendo la probabilidad de que el bot realice dicha tarea en n pasos, y a nosotros nos interesa que lo haga en n pasos o menos. Por tanto la fila 7 de la matriz será distinta, indicándole que una vez que estemos en el punto 7 ya hemos llegado al destino. Esto se consigue rellenando toda la fila con ceros, excepto la componente (7,7) que se le dará el valor de 1, representando que una vez que estamos en el punto 7 la probabilidad de permanecer en tal punto es del 100 %, por tanto, hemos llegado al destino. De esta manera, ya podremos consultar en la matriz la probabilidad de que un bot llegue a cualquier punto en n pasos:

$$p = M^n[x, y]$$

A modo de ejemplo, en la tabla 5.1 se muestran las probabilidades de alcanzar el punto destino(7) en un ciclo y en diversas situaciones, según el valor de IH y el punto el partida. Si quisieramos calcular las probabilidades para varios ciclos, bastaría con multiplicar dichas probabilidades por el número de ciclos que queramos comprobar.

IH	Punto de partida		
	20	21	22
0.1	0	0	0
0.2	0.0156	0.0313	0.0563
0.3	0.0297	0.0638	0.0982
0.4	0.0498	0.0913	0.1368
0.5	0.0692	0.1173	0.1673
0.6	0.0895	0.1408	0.1945
0.7	0.1096	0.1630	0.2184
0.8	0.1297	0.1840	0.2404
0.9	0.1496	0.2042	0.2608
1	0.1691	0.2236	0.2800

Tabla 5.1: Probabilidades de alcanzar el destino en un ciclo de 10 segundos, desde las tres posiciones de salida (ver figura 5.2) con diferentes IH's.

Ya conocemos con qué probabilidad un bot, según su IH y su punto de salida, encontrará la bandera. En la siguiente sección se modelará como conocer el tiempo que tardan el resto de bots en resolver el problema una vez que el primero ya la ha encontrado, que será dependiente de cuál de los bots la encuentre y de donde estén sus compañeros en ese momento.

5.3.2. Estructura del equipo según su IH

En cuanto al segundo factor presentado, podemos analizar las posibles combinaciones de los valores de IH de los tres bots y calcular el tiempo que tardarían en resolver el problema una vez que el primer bot haya encontrado la bandera. Esto último se ha presentado en el punto anterior.

En la figura 5.3 podemos ver gráficamente el ejemplo de un equipo de bots con diferentes IH's (0.25, 0.55 y 0.88). Las zonas marcadas en rojo corresponden al tiempo de exploración de cada bot, dónde les es imposible recibir los avisos de sus compañeros y tendrán que pasar al estado 'escuchando' para poder recibir nuevas comunicaciones o leer los avisos pendientes. Dicho estado está representando por las zonas de color verde.

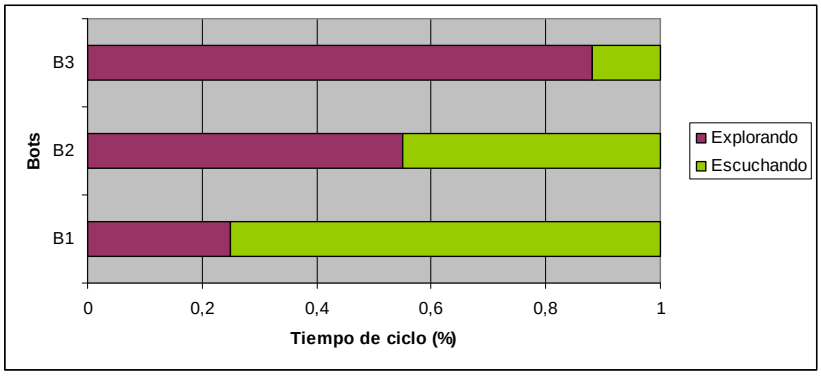


Figura 5.3: Representación gráfica de un tiempo de ciclo para un equipo de bots con diferentes IH's

Nos interesará que una vez que un bot encuentre la bandera, sus otros dos compañeros se encuentren en el estado 'escuchando' para acudir inmediatamente y dar por exitosa la resolución de la tarea. La eficiencia del sistema dependerá entonces de cuál sea el bot que encuentre la bandera, lo que podemos formular de la siguiente manera:

$$Bandera\, encontrada\, por =$$
$$\begin{cases} Bot\,1 & T_t = (IH_3 \cdot T_c + MAX(T_{nav_2}, T_{nav_3})) \cdot n_{ciclos} \\ Bot\,2 & T_t = (IH_3 \cdot T_c + MAX(T_{nav_1} - IH_3 \cdot Tc, T_{nav_3})) \cdot n_{ciclos} \\ Bot\,3 & T_t = (MAX(IH_2 \cdot T_c, T_{enc_3)} + MAX(T_{nav_1} - IH_3 \cdot Tc, T_{nav_2})) \cdot n_{ciclos} \end{cases}$$

$$\text{donde } IH_1 \leq IH_2 \leq IH_3$$

Llamamos T_t al tiempo en el cual se resuelve el problema, IH_i es el índice de hiperactividad del bot i, T_c es el tiempo de ciclo aplicado en el sistema, $Tnav_i$ es el tiempo que tarda el bot i en ir desde su posición actual hasta la bandera objetivo, T_{enc_i}es el tiempo que le cuesta al bot i encontrar la bandera aleatoriamente y n_{ciclos} es el número de ciclos necesarios para que alguno de los bots encuentre la bandera. Podemos deducir que la forma de minimizar el tiempo total de resolución del problema, es que el bot 3 encuentre la bandera en el primer ciclo y antes de que el bot 2 pase al estado 'escuchando', lo cual podemos ver gráficamente en la figura 5.4.

5.3.3. Evolución genética

Una vez que los bots estén operativos y la forma de la solución óptima analítica modelada, implementaremos un algoritmo genético que irá evolucionando bots y probando distintas soluciones, para ver cual se ajusta más a la óptima.

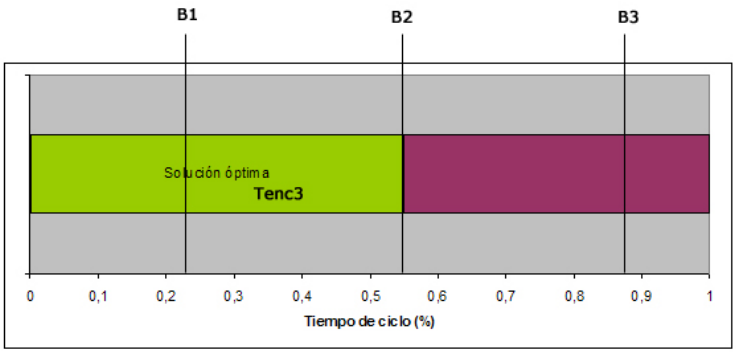


Figura 5.4: Solución óptima del problema siSosiG, donde el bot 3 encuentra la bandera antes de que el bot 2 cambie de estado, lo que minimiza el tiempo de resolución del problema

El genotipo de cada bot vendrá dado por un número real que se corresponderá con el IH definido anteriormente. Cada bot tiene su propio genotipo, por tanto estamos resolviendo el problema mediante la aproximación aclonal. En el proceso de evaluación emplearemos la función fitness, que viene definida como el tiempo que tardan todos los bots en llegar a la bandera, por tanto su valor es inversamente proporcional a la eficiencia del equipo.

$$\text{Fitness} = min(T_{total})$$

Para cada simulación, limitamos el tiempo máximo a 35 segundos, por lo que cualquier equipo de bots que sobrepase dicho tiempo, será puntuado con una fitness de 35, que será considerado como objetivo incumplido.

Como el movimiento por el mapa de nuestros bots es aleatorio, la combinación de los mismo genotipos para una población, no tiene porque generar lo mismo resultados. El objetivo de la evolución será entonces, encontrar aquella combinación o combinaciones de genotipos que resuelvan el problema de forma eficiente para el mayor porcentaje elevado de ocasiones.

5.4. Resultados

Después de casi 2900 generaciones, obtenemos un listado ordenado de las mejores fitness, junto a los genotipos que han generado dichas soluciones, los cuales podemos ver en la tabla 5.2.

De las 2900 combinaciones de genotipos, el 60.88 % resuelve el problema en menos de los 35 segundos máximos dados, los cuales podemos subdividir en distintos umbrales, como podemos ver en la tabla 5.3.

5.4.1. Análisis de Resultados

Lo que nos analizar en claro de este experimento, son las estructuras de bots (combinaciones de genotipos) que nos proporcionen un mayor porcentaje de éxito en cada uno de los umbrales expuestos anteriormente (35, 25, 15 y 10 segundos). Puesto que son 2900 simulaciones, necesitaremos un método analítico que nos permita sacar las conclusiones correctas.

Si nos encontráramos en el caso de un agente único, las conclusiones serían fáciles, ya que bastaría con calcular qué valores o rango de valores del genotipo aparecen más frecuentemente para cada uno de los umbrales. Pero en el modelo multiagente, necesitamos mantener la estructura y la conexión entre los diferentes bot.

Fitness	IH - Bot 1	IH - Bot 2	IH - Bot 3
7.187	0.32	0.48	0.56
7.312	0.19	0.41	0.41
7.766	0.02	0.08	0.26
7.766	0.15	0.17	0.29
7.828	0.36	0.46	0.48
7.844	0.15	0.17	0.47
7.875	0.19	0.27	0.93
7.89	0.26	0.34	0.36
8.25	0.35	0.49	0.99
8.25	0.2	0.58	0.66
8.313	0.32	0.48	0.56
8.375	0.04	0.22	0.7
8.437	0.32	0.48	0.56
8.828	0.41	0.45	0.47
8.875	0.34	0.58	0.98

Tabla 5.2: 15 mejores resultados del experimento siSosiG

Porcentaje	Resuelve en menos de
60.88 %	35 segundos
42.46 %	25 segundos
15.56 %	15 segundos
2.59 %	10 segundos

Tabla 5.3: Resultados siSosiG

Regresión logística

La regresión logística es un modelo de regresión para variables dependientes o de respuesta binomialmente distribuidas. El objetivo primordial que resuelve esta técnica es el de modelar cómo influye en la probabilidad de aparición de un suceso, habitualmente dicotómico, la presencia o no de diversos factores y el valor o nivel de los mismos. En nuestro caso, el suceso será el umbral de segundos que los bots deben superar y los factores serán las distintas estructuras y combinación de genotipos.

La expresión de la regresión logística viene dada por:

$$p(x) = \frac{1}{1+e^{-(B_0+B_1 \cdot x_1+\ldots+B_n \cdot x_n)}}$$

donde:

p: es la probabilidad de que ocurra el suceso

B_x: son los parámetros calculados por la regresión

X_i: son las entradas para las que calculamos la probabilidad, en nuestro caso los índices de hiperactividad de cada bot del equipo.

El propósito del análisis con la regresión logística nos permitirá en nuestro caso:

- Predecir la probabilidad de que se resuelva el problema por debajo de un determinado umbral (en segundos) para un determinado genotipo.

- Determinar qué variables (índices de hiperactividad) pesan más para aumentar o disminuir la probabilidad de tener éxito en la tarea.

Esta asignación de probabilidad de ocurrencia del evento a un cierto genotipo, así como la determinación del peso para cada una de las variables dependientes en esta probabilidad, se basan en las características que presentan los distintos genotipos simulados.

Para poder analizar correctamente los resultados, hemos ordenado los índices de hiperactividad de los bots en orden creciente. Estos nos permitirán calcular de forma coherente las medias y desviaciones típicas de cada uno de los índices, como podemos ver en la figura 5.4, lo que nos servirá para realizar unas nuevas simulaciones de comprobación de los resultados de la regresión.

Los resultados obtenidos con la regresión logística para los 2900 datos y los umbrales de fitness 35, 25, 15 y 10 los podemos ver en la tabla 5.5. En dicha tabla podemos ver tanto los coeficientes que se aplicarán a cada entrada (IH del bot), así como sus respectivos p-valores. Un coeficiente positivo, indica que un valor más alto de la entrada hará que el sistema tenga más posibilidad de tener éxito. Lo contrario ocurre con coeficientes negativos. El p-valor nos indica la relevancia, a la hora de calcular la probabilidad final de que un objetivo se cumpla, de cada uno de los coeficientes. En la regresión logística, un p-valor mayor que 0.05, indica que el coeficiente sobre el que se aplica no es significativo (marcados en rojo).

Bot	Media	Desv. Típica	Rango
B1 (bajo)	0.26	0.19	0.07 - 0.45
B2 (medio)	0.51	0.22	0.29 - 0.73
B3 (alto)	0.75	0.19	0.56 - 0.94

Tabla 5.4: Medias y desviaciones típicas para cada uno de los bots en el experimento siSosiG

Umbral	B1		B2		B3	
	Coef.	P-valor	Coef.	P-valor	Coef.	P-valor
35	1.00155	0.0	0.982621	0.0	0.184483	0.453
25	0.705250	0.003	1.01927	0.0	0.583642	0.023
15	0.742842	0.012	1.16226	0.001	0.996409	0.010
10	0.0120865	0.988	-0.046455	0.956	-2.18946	0.003

Tabla 5.5: Resultados obtenidos en el experimentos siSosiG con regresión logística

Con todos los datos calculados, queremos comprobar si la regresión ha sido efectiva y nos ha servido para configurar equipos de bots que resuelvan de forma más rápida el problema de encontrar la bandera, dentro de sus umbrales de tiempo. Para ello vamos a crear un equipo de bots para cada umbral, siguiendo las siguientes normas:

1. Si el p-valor nos indica que el coeficiente de un determinado bot no es significativo, tomaremos el valor medio del IH de dicho bot.

2. En caso de que el coeficiente sea positivo, y el p-valor indique que es relevante, cogeremos el IH más grande dentro de su rango.

3. Si nos encontramos con que el coeficiente es negativo y el p-valor nos revela que es significativo, cogeremos el IH más pequeño dentro del rango obtenido con la media y la desviación típica.

Por tanto, siguiendo esta reglas obtenemos los distintos equipos para cada uno de los umbrales que podemos ver en la figura 5.6.

Umbral	B1	B2	B3
35	0.45	0.73	0.75
25	0.45	0.73	0.94
15	0.45	0.73	0.94
10	0.26	0.51	0.56

Tabla 5.6: Equipos de bots creados para probar los resultados de la regresión logística

Con los equipos ya creados, simularemos cada uno de ellos 2900 veces, esta vez sin un proceso genético puesto que estamos ante genotipos que no van a cambiar. El objetivo de estas simulaciones es ver si los resultados que nos ha proporcionado la regresión logística son válidos y nos ha permitido obtener unas configuraciones de bots, que son más eficientes a la hora de afrontar la tarea en sus diferente umbrales. Los resultados de las simulaciones los podemos ver en la tabla 5.7, donde marcamos con verde los que han superado a simulación general y en rojo los que no. Gráficamente podemos ver los resultados en la figura 5.5.

Umbral	General	Equipo 35	Equipo 25-15	Equipo 10
35	60.88 %	71.11 % (*)	67.20 %	64.52 %
25	42.46 %	53.45 % (*)	51.33 %	43.67 %
15	15.56 %	20.18 %	20.98 %	10.62 %
10	2.59 %	0.76 %	0.49 %	4.75 % (*)

Tabla 5.7: Resultados de las simulaciones de los equipos de bots que hemos generado a partir de la regresión logística. (*) Mejores resultados para el umbral

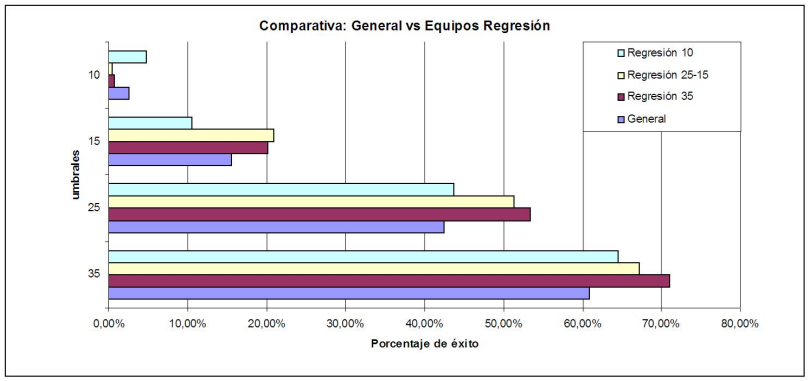


Figura 5.5: Resultados gráficos de las simulaciones de los equipos de bots que hemos generado a partir de la regresión logística

5.4.2. Interpretación

En primer lugar vamos a presentar las posibles razones de las configuraciones de los equipos que nos ha dado la regresión logística:

- Equipo umbral 35: Resolver el problema en menos de 35 segundos no es algo complicado, como podemos deducir de los resultados de la simulación general, en la que casi un 61 % de los equipos los consiguen. Como el tiempo de ciclo son 10 segundos, tenemos 3 ciclos y medio para llegar a nuestro objetivo. Con bots con un IH bajo, estamos perdiendo capacidad

de exploración y como tenemos varios ciclos (los bots que encuentren la bandera tienen tiempo para avisar a sus compañeros) nos interesa tener bots bastante activos, que busquen la bandera y avisen a sus compañeros. El bot 3 ya es activo de por sí, por lo que nos dará igual su valor siempre que nos movamos en sus rangos de valores.

- Equipo umbral 25-15: En este caso tenemos menos ciclos para resolver el problema, por lo que necesitaremos bots todavía más activos, aún a riesgo de tener problema de comunicación. Por ese motivo cogemos el valor más alto del B3, al igual que en B1 y B2.
- Equipo umbral 10: Superar este umbral es algo muy complicado, un porcentaje muy bajo de equipos lo consigue. En este caso todo depende del bot más activo (B3), puesto que un valor muy alto le supondrá tener que encontrar la bandera y avisar a los demás. En el caso de que éste no la encuentre y lo haga uno de sus compañeros, es posible que no escuche la transmisión puesto que sigue explorando el terreno. Por tanto, nos interesa un valor bajo de B3 para que cualquiera de los bots que componen el equipo sea capaz de recibir la transmisión y llegar hasta la bandera en menos de 10 segundos.

Los resultados obtenidos en la primera simulación, son inferiores a casi todos los obtenidos por los nuevo equipos. Esto es debido a que la evolución genética introduce en muchas ocasiones genotipos con poco sentido, como puede ser un equipo de bots con los valores B1=0.01, B2=0.05, B3=0.06.

Estamos ante un entorno muy aleatorio, puesto que hemos dejado mucha libertad en los movimientos de los bots, y la función fitness no castiga ni premia ningún comportamiento, solo es dependiente del tiempo en el que se resuelve la tarea. Esto se hace latente en los resultados, puesto que para el mismo equipo de bots, rara vez se obtiene el mismo resultado. La aleatoriedad del experimento no lo hace inválido, si no que introduce un factor de peso que marcará los resultados. Podemos dar el experimentos como válido, demostrando que si queremos superar alguno de los umbrales, con el uso de los equipos obtenidos con la regresión logística, tendremos mas posibilidades de éxito.

Capítulo 6

Experimento 2 - Pathwalker Bot

Wanderer, your footsteps are
the road,and nothing more;
wanderer, there is no road,
the road is made by walking.

(Antonio Machado)

El objetivo de este experimento es demostrar que un grupo de bots totalmente iguales (aproximación clonal), usando técnicas basadas en algoritmos bioinspirados tipo swarming, son capaces de resolver ciertas tareas en su entorno.

Para poder quitar ese grado de aleatoriedad que tenía el experimento anterior vamos a introducir: (1) una implementación del bot con un movimiento menos aleatorio, gracias a la feromona, y (2) una función fitness más específica, lo que nos permitirá reducir la aleatoriedad de los resultados cuando se repite el experimento con los mismos genotipos.

6.1. Descripción del problema

Nos encontramos en un mapa de captura de bandera. El objetivo de nuestro equipo de bots es en primer lugar encontrar una de las banderas (la enemiga o la del propio equipo) y posteriormente normalizar un camino entre ellas, a través de un rastro que los bots dejan a su paso, como podemos ver en la figura 6.1, para que todos los demás bots del equipo puedan encontrarlas también. En el momento que el ultimo bot encuentre una de las dos banderas, daremos el experimento como válido. Es este caso no habrá comunicación directa entres los miembros del equipo, pero podrán guiar su movimiento gracias al rastro de una "feromona".

Tal y como explicamos en el capítulo anterior, los mapas de Unreal Tournament, tienen definidos una serie de puntos de navegación (NavPoints) que el bot puede emplear para moverse por ellos. Inicialmente, los bots serán creados en varios puntos de navegación predefinidos, etiquetados como "PlayerStart-X". Deberán encontrar el punto de navegación en el cual se encuentra la bandera enemiga. Para ello, se moverán entre los puntos de navegación, calculándose donde tienen que realizar el próximo movimiento, realizando siempre un movimiento a los puntos de navegación adyacentes.

Vamos a crear un modelo bioinspirado, asemejándonos con el mecanismo que tienen las hormigas para crear caminos. Uno de los retos de trabajar con algoritmos de hormigas, es que estos

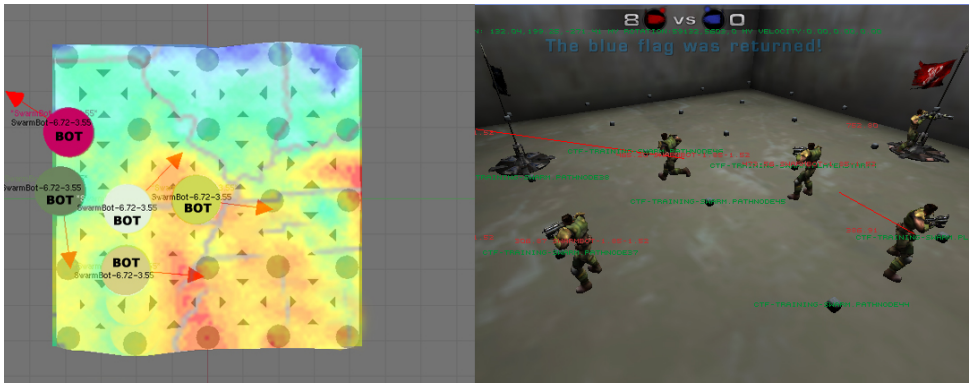


Figura 6.1: Ejemplo del rastro de feromona que los bots dejan a su paso

incluyen una serie de parámetros que regulan el comportamiento de nuestro sistema. Estos parámetros regulan la influencia de la selección de los distintos caminos por parte de las hormigas, pudiendo:

- Seguir: El bot toma un camino ya creado y conocido.
- Explorar: El bot sigue un camino desconocido, con la posibilidad de encontrar una solución al problema.

Como podemos observar, se nos plantea de nuevo el dilema de exploración-explotación explicado en el capítulo anterior. Ambos factores tendrán relación directa con el valor de la feromona en los distintos puntos y con los parámetros que marcarán la rapidez de crecimiento y evaporación de la feromona.

6.2. Implementación

Para simbolizar el comportamiento de un grupo de hormigas, vamos a crear un tabla con los diferentes puntos de navegación del mapa, asignando un valor de feromona a cada NavPoint. Inicialmente el valor de la feromona será vacío, lo que nosotros representaremos con el valor numérico 1 por cuestiones de implementación. La feromona puede variar por dos motivos:

1. Un bot pasa por el punto de navegación: Esto hará que el valor de la feromona aumente. El aumento dependerá de una función matemática que definiremos más adelante.
2. Evaporación de la feromona: Tal como en el mundo natural, el entorno hace que la cantidad de feromona depositada en punto se vaya diluyendo con el paso del tiempo, si dicho punto no es visitado por ninguna hormiga. Al igual que en el punto anterior, una función matemática se aplicará cada cierto tiempo para simular esa pérdida de feromona.

A la hora de moverse por el mapa, los bots tendrán en cuenta la feromona de sus puntos adyacente. El siguiente movimiento de cada bot vendrá decidido por un algoritmo aleatorio con pesos, por tanto tendrá más posibilidades de ser visitado un NavPoint con un mayor nivel de feromona. Se empleará un algoritmo de selección por ruleta para calcular el punto destino.

Se ponderará a 100 la suma de los valores de la feromona de todos los puntos de navegación adyacentes, y a cada NavPoint se le asignará una porción de ruleta, que corresponderá con su valor

de feromona ponderado. Seguidamente se lanza el selector y nos da el punto destino. Evidentemente tiene mayor probabilidad el NavPoint con mayor activación de feromona.

Uno de los aspectos más importantes del sistema es mediar como el nivel de feromona va cambiando a lo largo de la partida. Esto vendrá determinado por el número de bots que pasen por cada NavPoint, así como el proceso que diluye dicha feromona. Como hemos dicho anteriormente, vamos a calcular la velocidad con la que la feromona aumenta o se diluye, a través de dos parámetros incluidos en las funciones matemáticas de incremento y decremento.

A continuación podemos ver la función de incremento de feromona:

$$I(t_i) = A(1 - e^{-t_{i/\tau}}) + 1$$

donde:

A: Límite superior de feromona. Es el valor donde la feromona satura, es decir, llega un momento que por muchos bots que pasen, la feromona no sigue creciendo. Nosotros hemos puesto en 30 el valor límite.

τ: Parámetro que marca la rapidez de crecimiento, inversamente proporcional a su valor

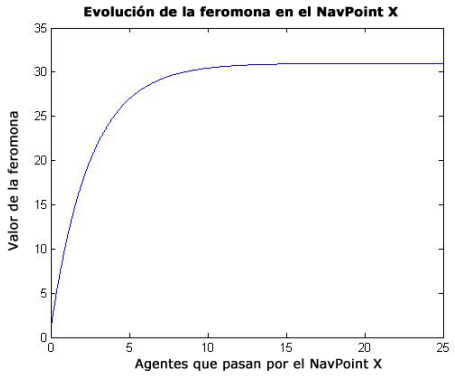


Figura 6.2: Función de incremento con $\tau = 2{,}5$

En la figura 6.2, podemos ver un ejemplo de como se comportaría el crecimiento de la feromona con un valor de τ de 2.5. Dicha gráfica representaría la evolución de un único Navpoint. Podemos ver que, aproximadamente, cuando 14 bots pasen por encima de dicho punto, la feromona saturará. Eso suponiendo que no tuviéramos ningún función que diluya la feromona, lo cual, no es nuestro caso.

Cuando en un punto de navegación no pase ningún bot, cada segundo se ejecutara una función matemática que hará que la feromona vaya perdiendo fuerza. Esta función depende del parámetro ε:

$$I(t_d) = A(e^{-t_d/\varepsilon}) + 1$$

donde:

A = Límite superior de feromona, igual que en la fórmula anterior.

ε = Parámetro que marca la rapidez de decrecimiento, inversamente proporcional a su valor.

En la figura 6.3 podemos ver como un NavPoint con la feromona saturada, se diluye con el paso el tiempo hasta llegar a su límite inferior, 1.

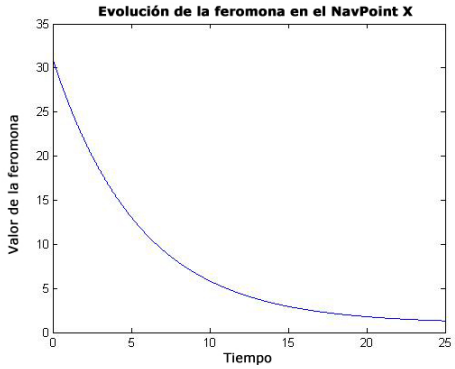


Figura 6.3: Función de decremento con $\varepsilon = 5{,}5$

Una vez con ambas fórmulas en funcionamiento, aparte de las gráficas anteriores, podremos obtener durante la simulación una gráfica que sea una mezcla de ambas. Esto es debido al paso intermitente de bots por los puntos de navegación, que harán que el valor de la feromona vaya aumentando o disminuyendo de una manera no lineal.

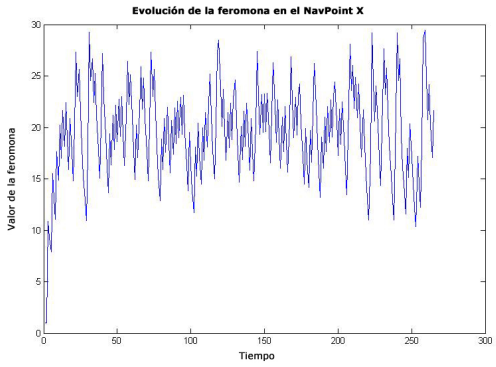


Figura 6.4: Ejemplo del valor de la feromona de un NavPoint con $\tau = 2{,}5$ y $\varepsilon = 5{,}5$

En la figura 6.4, vemos un ejemplo de cómo se comportará el valor de la feromona en un determinado NavPoint para los valores dados de tau y epsilon.

A la hora de aplicar estas dos funciones simultáneamente hay que tener en cuenta el valor de la feromona en el instante en el que se aplica cada una.

Como podemos ver en la figura 6.5, cuando en un determinado momento tengamos que cambiar la función a aplicar (punto A), deberemos conocer el valor de tiempo que provocaría que nuestra función a aplicar tuviera el valor de feromona actual (punto B). Una vez que tenemos dicho valor lo incrementaremos en una unidad y calcularemos en siguiente valor de feromona (punto C). Dependiendo de la función que queramos aplicar deberemos calcular T_i para la función de incremento, siendo I el valor actual de la feromona, la cual podemos se define de la siguiente manera:

$$Ti = -ln(\tfrac{1-I}{A} + 1) \cdot \tau$$

o T_d, en el caso del decremento:

$$Td = -ln(\tfrac{I-1}{A}) \cdot \epsilon$$

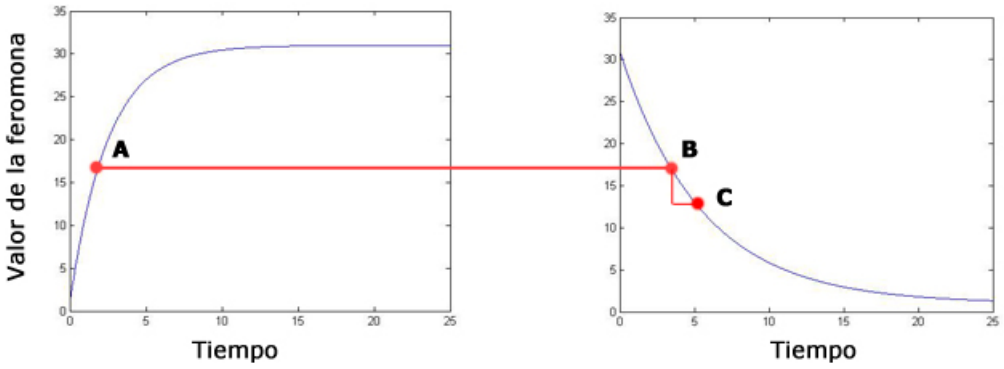


Figura 6.5: Cálculo del siguiente valor de la feromona al cambiar la función a aplicar

6.3. Estudio analítico

Como podemos ver en el apartado anterior, el valor de la feromona es lo que va a marcar en mayor medida el movimiento de nuestros bots. Nos interesa entonces, una vez que el primer bot ha encontrado la ruta entre banderas, que en el camino existente en dicha ruta el valor de la feromona esté siempre saturado, tal y como podemos ver en la figura 6.6, representando el valor medio de la feromona en el camino entre banderas con respecto al tiempo: $ph_{path}(t)$. Esto provocaría que todos los agentes que todavía no están por ese camino, se sintieran fuertemente atraídos por él, abandonando su exploración propia en curso.

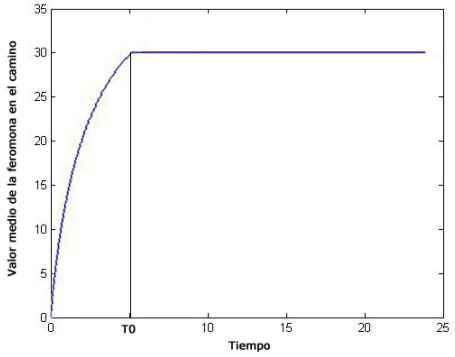


Figura 6.6: Escenario ideal del valor de la feromona, donde T_0 es el tiempo que tarda el primer bot en encontrar el camino entre banderas. El eje Y representa el valor medio de la feromona en todos los puntos del camino.

Hemos descrito la situación ideal, algo imposible que se de en la realidad, puesto que los valores de τ y ϵ que dieran lugar al estado descrito provocarían que se crearan otros caminos con un alto valor de feromona(ya que no tenemos manera alguna de indicar que solo queremos que se alimente el camino entre banderas). Por tanto, tendremos que buscar el equilibrio entre la creación de caminos y la realimentación de los ya creados, lo cual será totalmente dependiente de τ y ϵ. Puesto que no podemos llegar a obtener la situación ideal descrita, tendremos que intentar alcanzar una solución que sea lo más parecida posible, es decir, una solución que se ajuste lo máximo posible al ideal. Este ajuste es lo que nos dirá lo buena o mala que es una solución propuesta. Una función dependiente de la feromona y el tiempo se estará ajustando al estado óptimo cuando el valor de la

feromona vaya en aumento o se mantenga igual y se estará alejando cuando el valor decaiga. Por tanto, nuestra función de ajuste quedaría de la siguiente manera:

$$a(t) = \begin{cases} En\,ajuste, & ph_{path}(t_{i-1}) \leq ph_{path}(t_i) \\ En\,desajuste, & ph_{path}(t_{i-1}) > ph_{path}(t_i) \end{cases}$$

Donde se relacciona el valor correspondiente a la situación en ajuste y a la de desajuste.

En la figura 6.7 podemos ver gráficamente como se comporta la función de ajuste, representando en verde los intervalos de tiempo dónde la solución esta ajustando, y en rojo donde no.

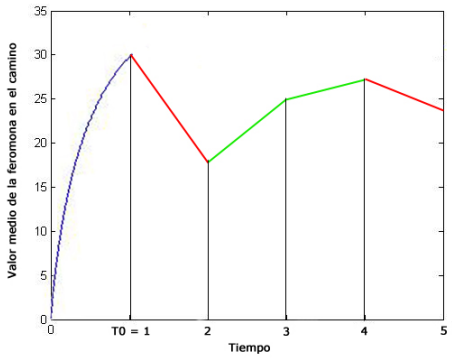


Figura 6.7: Función de ajuste para nuestro problema

Con la función de ajuste calculada analíticamente, necesitaremos ahora aplicar una evolución genética a nuestros equipos de bots para buscar soluciones cada vez más ajustadas a la solución óptima.

6.3.1. Evolución genética y Fitness

Hemos visto que el éxito de un equipo de bots a la hora de llegar todos a la bandera de manera rápida, dependerá casi exclusivamente de los valores de τ y ϵ. Éstos, según sus valores, harán que los caminos se fortalezcan de manera más rápida, lenta o incluso que no se fortalezca ningún camino o que todas las rutas sean caminos. Sabemos que lo que nos interesa es que se fortalezca la ruta entre las dos banderas y las demás posibilidades de rutas, no sean caminos fortalecidos. Deberemos encontrar los valores de los parámetros τ y ϵ que hagan eso posible.

Como en cualquier algoritmo genético la función fitness apropiada es la clave para obtener buenos resultados. En nuestro caso queremos que nuestra función de adecuación contemple los siguientes factores:

1. El tiempo total de la partida.
2. El instante en el que el primer bot encuentra alguna de las banderas.
3. El grado de fortalecimiento del camino entre banderas una vez que el primer bot ha encontrado la bandera, es decir, la función de ajuste calculada en el apartado anterior.

Queremos una función resistente a los factores aleatorios, por eso introducimos el segundo punto, ya que lo que le cuesta al primer bot encontrar la bandera puede considerarse aleatorio y de esta forma, podemos quitar ese tiempo de la función fitness. Si tenemos dos escenarios con dos equipos de bots del siguiente modo:

1. Tiempo total de partida= 15 segundos. Tiempo en encontrar por primera vez la bandera = 2 segundos.
2. Tiempo total de partida= 20 segundos. Tiempo en encontrar por primera vez la bandera = 15 segundos.

Si solo consideráramos el tiempo total de la partida, como en el experimento anterior, el primer equipo sería mejor a la vista de nuestro algoritmo genético. Pero realmente, el segundo equipo esta demostrando un mejor comportamiento, puesto que una vez encontrada la bandera por primera vez, el tiempo que le ha costado al equipo fortalecer el camino y hacer que todos los bots lleguen a su destino es mucho menor.

La función fitness quedaría de la siguiente manera:

$$F = 2 \cdot (T_t - T_0) + \sum_{T_0}^{T_t} a(t)$$

dondeT_t es el tiempo total de la partida. El tiempo de simulación máximo lo hemos fijado en 35 segundos, por lo que si cualquier equipo de bots llega a ese límite, su fitness valdrá 0. El parámetro T_0, es el instante de tiempo dentro de la simulación donde un bot encuentra la bandera por primera vez. La función de ajuste, ya que nuestro objetivo es minimizar el valor de la función fitness, quedaría de la siguiente forma:

$$a(t) = \begin{cases} 0, & ph_{path}(t_{i-1}) \leq ph_{path}(t_i) \\ 1, & ph_{path}(t_{i-1}) > ph_{path}(t_i) \end{cases}$$

En el mapa en el que se realiza la simulación, existen un número N de NavPoints que definen el camino entre las dos banderas. Nos interesará que una vez que se encuentre una de las dos banderas por primera vez, el camino entre ambas se fortalezca y nunca se diluya. Con esta función de ajuste estamos penalizando cada vez que se produce un descenso de feromona en el camino entre banderas.

Con la función fitness definida, la efectividad de un equipo de bots es inversamente proporcional al valor de dicha función, por lo que siempre buscaremos fitness mínimas. Podemos ver en la fórmula de la función, que hemos introducido un término multiplicativo en uno de los factores. Esto es así, puesto que en simulaciones de prueba observamos que el valor del sumatorio de la función de ajuste casi siempre es mayor que el factor de la diferencia de tiempos en un 75-110 %. Como queremos que ambos factores pesen lo mismo a la hora de calcular el valor de la fitness, introducimos el termino 2 multiplicando al primer factor.

6.4. Resultados

Después de casi 1600 generaciones, obtenemos un listado ordenado de las mejores fitness, junto a los valores de τ, ε y de cada uno de los términos que intervienen en la función de adecuación, los cuales podemos ver en la tabla 6.1.

De los 1600 resultados el 80,14 % resuelve el problema en menos de los 35 segundos límite fijados para el experimento, distribuidos de la manera que podemos ver en la tabla 6.2.

6.4.1. Análisis de Resultados

Lo que nos interesa sacar en claro de este experimento, es saber qué combinaciones de parámetros τ y ϵ nos dan unos mejores resultados, es decir, qué genotipos generan una mejores fitness. En

Fitness	τ	ϵ	T_0	T_t	$ajuste$
2.656	3.77	7.17	1.63	3.28	2
3.422	1.39	5.24	1.09	3.52	4
3.485	7.42	7.72	2.17	4.66	4
3.516	5.81	7.50	2.16	4.67	4
3.688	0.93	5.12	2.17	4.86	5
3.766	1.22	4.83	1.03	3.30	4
3.766	3.61	9.55	2.05	4.31	4
3.953	4.02	10.14	3.41	5.86	5
3.953	9.33	3.40	2.97	5.42	5
4.125	9.43	9.94	6.36	8.99	5
4.203	6.17	5.69	1.13	3.83	6
4.266	9.20	9.15	1.67	4.44	6
4.312	1.27	5.80	3.67	6.48	6
4.453	1.09	4.31	1.91	4.86	7
4.531	3.70	0.92	3.28	6.31	7

Tabla 6.1: 15 mejores resultados del experimento Swarm

Porcentaje	Fitness menor que
80.14 %	50
79.95 %	45
79.37 %	40
76.74 %	35
72.69 %	30
65.62 %	25
53.92 %	20
36.89 %	15
15.23 %	10
1.61 %	5

Tabla 6.2: Resultados del experimento Swarm

este caso sólo tenemos dos entradas a analizar (τ y ϵ) y una salida para poder medir la calidad de la solución (función fitness). Por definición de nuestra función fitness, nos interesarán unos valores mínimos como resultado de esta función.

Para poder conclusiones sobre los resultados, calcularemos para cada genotipo el cociente entre ϵ y τ, lo que llamaremos coeficiente de tendencia al crecimiento (C_{tc}) del sistema. Valores altos de este coeficiente nos indicarán que nuestro sistema, por la constucción de sus funciones de crecimiento y decrecimiento de la feromona, tiende a mantener unos valores altos en todos los puntos de navegación. Una vez calculados los coeficientes, representamos gráficamente dichos coeficientes con respecto a su valoración fitness (figura 6.8).

En la figura 6.8, podemos observar tres zonas donde un rango de coeficientes nos generan un mejor o peor resultado:

- *Zona 1 ($C_{tc}\epsilon$[0.05-0.90]):* En este rango de coeficientes en líneas generales podemos ver que los resultados no son los deseados, obteniendo unos valores de adecuación bastante altos.
- *Zona 2 ($C_{tc}\epsilon$[0.90-1.20]):* Nos encontramos ante el rango donde se han obtenido los mejores resultados.

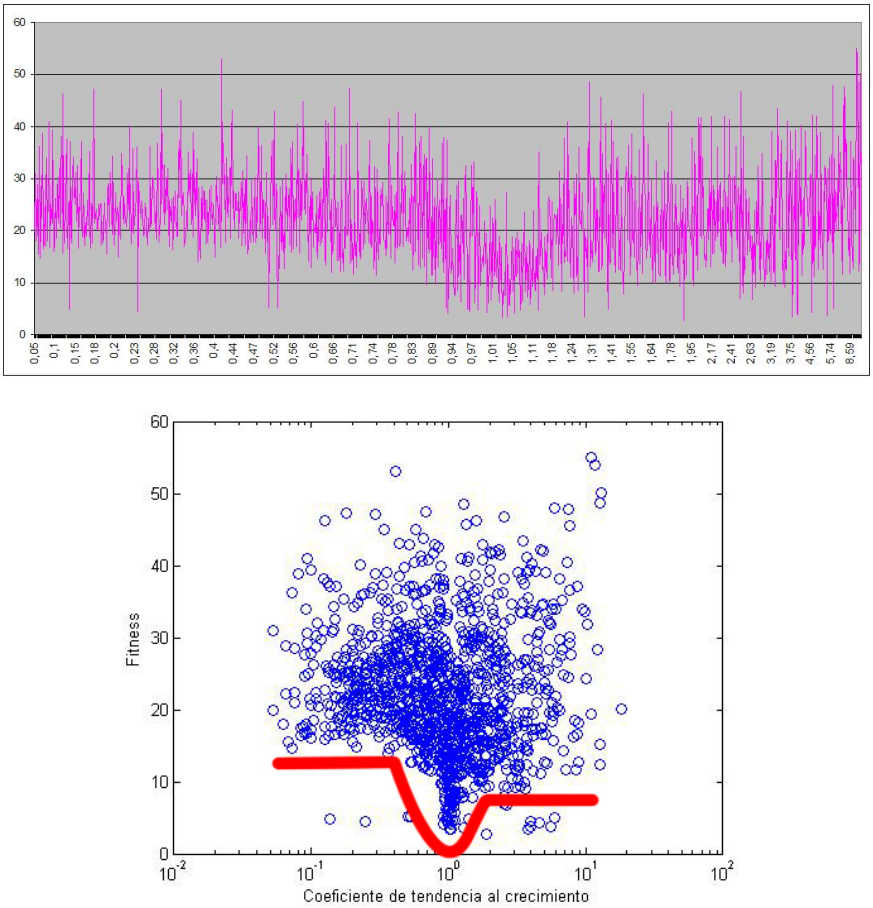


Figura 6.8: Representación gráfica de los coeficientes de tendencia el crecimiento con respecto a la función fitness. La primera gráfica esta construida sin escala y representa un valor por cada muestra, y en la segunda podemos ver todo el rango de valores bajo un eje de abcisas logarítmico.

- *Zona 3 ($C_{tc}\epsilon$[1.20-18):* Aquí podemos ver unos resultados bastante dispersos, donde nos encontramos situaciones de todo tipo, aunque en proporción nos encontramos mejores soluciones que en la primera zona.

A la vista de los resultados nos interesará contar con un coeficiente que se encuentre en la Zona 2. En la tabla 6.3 vemos las medias de los valores que nos devuelve la función fitness con respecto a la zona de rangos en la que nos encontramos.

Zona	Media	Muestras
General	22.09	1595
1	24.01	707
2	15.66	235
3	22.11	653

Tabla 6.3: Relación de las medias de las fitness con las zonas de rangos de los coeficientes

6.4.2. Interpretación

En el análisis de resultados podemos ver que en algunas zonas de distintos rangos de coeficientes, encontramos resultados muy variados. Por ejemplo, en la zona 1 podemos ver que por lo general obtenemos resultados que resuelven el problema de una manera poco eficiente, pero también podemos encontrar algunos coeficientes puntuales que resuelven el problema con una buena fitness. Podemos atribuir estas distorsiones al componente aleatorio del sistema, por lo que nos tendremos que quedar con la visión general de los resultados y despreciar estas anomalías.

A la vista de los resultados obtenidos, podemos ver que la zona 2 posee unos coeficientes de tendencia al crecimiento que se encuentran entre el rango 0.90-1.20, es donde se obtienen unos mejores resultados. Además vemos que en la útlima zona(1.20-18) se obtienen también buenos resultados, aunque nos encontramos también algunos tan malos como en la zona 1.

La zona 1 esta formada por coeficientes bajos, lo que hace que la evaporación de la feromona sea más rápida que la creación de la misma. Esto supone que es muy difícil que se fortalezcan caminos entre banderas, pues el factor de evaporación va a ser siempre elevado. Los casos de buenas fitness que nos hemos encontrado en esta zona, son debidos a que el azar ha hecho que el equipo de bots encuentre rápidamente el camino y entre todos han conseguido que la creación del camino tenga más peso que la evaporación del mismo, pero por normal general vamos a obtener resultados poco eficientes en esta zona.

Para obtener buenos resultados, nuestros coeficientes deberán ser elevados. Esto implica que en nuestro problema es conveniente que las feromonas mantengan un valor alto y prevalezca la creación de caminos con respecto a la disolución de estos. Con este razonamiento podemos encontrar sentido a la disparidad de resultados en la zona 3, puesto que si tenemos un coeficiente elevado es posible que todos los bots enseguida se encaucen en en camino entre banderas, pero también puede suceder que cada uno cree su propio camino, tardando más en encontrar el correcto. El azar será el que marcará estas situaciones.

Dentro de nuestro dilema E-E particular planteado para este problema, donde estábamos buscando el equilibrio entre seguir y explorar, podemos concluir que el equilibrio se encuentra inclinado hacia el comportamiento de seguir los caminos ya creados.

Capítulo 7

Conclusiones

Como hemos contado al principio de esta memoria, existió una fase previa a la realización del proyecto que consistió en la documentación y formación sobre las herramientas y plataformas que se iban a emplear para realizar nuestros experimentos. Esto nos permitió ver si tanto Unreal Tournament 2004 como Pogamut, nos iban a permitir realizar nuestros estudios sin apenas limitaciones. UT2004 es de código abierto y es empleado en el concurso BotPrize, además cuenta con las librerías de Pogamut, por lo que parecía una buena opción abrir nuestras líneas de investigación sobre ellos.

La plataforma Pogamut como se comenta en la sección 3.1, es un conjunto de librerías que permiten establecer la comunicación con el videojuego Unreal Tournament 2004, sin necesidad de aprender a manejar su lenguaje de scripting (UnrealScript), ni su sistema de comunicación, ya que con estas librerías se pueden crear bots del juego de manera muy sencilla empleando Java. Durante la realización del proyecto se actualizó de la versión 2.0 a la 3.1, la cual es bastante más completa. Aun así, nos encontramos algunas funciones sin documentar e incluso sin implementar. De todas manera una vez que se adaptó el sistema a nuestros experimentos, se pudo comprobar cómo los pasos para construir un bot son bastante básicos al tener únicamente cuatro clases principales: (1) UT2004BotRunner que se encargar de lanzar el thread del bot; (2) PogamutException que maneja todas las excepciones que puedan surgir en el código; (3) UT2004BotModuleController , que contiene el controlador principal del juego; y (4) WorldView que controla los eventos que se producen en el entorno del juego. Por ello, manejándonos con esas clases podremos obtener un agente sin muchas complicaciones, teniendo mayores dificultades a la hora de buscar comportamiento más rebuscados. Unreal Tournament 2004 es un buen entorno donde realizar las simulaciones, puesto que nos ofrece un mundo bastante reactivo, donde podemos percibir muchos estímulos y a su vez reaccionar antes ellos con múltiples acciones. Las principales limitaciones que hemos encontado en la plataforma UT2004 + Pogamut son las siguientes:

- UT2004 es un entorno en tiempo real, lo que supone que, a pesar de que el flujo del tiempo puede ajustarse, no existe una opción "correr a la máxima velocidad posible", lo cual sería muy útil para reducir el tiempo necesario para las evaluaciones del algoritmo genético.
- Un incremento en la velocidad estándar del juego provocara fallos en el comportamiento de los bots, ya que no se terminan de ejecutar todas las instrucciones que determinan su comportamiento, y se producen por tanto resultados erróneos.
- Los tiempos de ciclo en UT2004 son irregulares. Pese a que es posible configurar manualmente cada cuánto tiempo *GameBots* debe ejecutar sus comandos de acción (el cual viene predefinido a 0.25 segundos, es decir, 4 acciones por segundo), la mala gestión por parte de

Pogamut provoca alternancias entre dos valores (siendo el más habitual un tiempo de ciclo irregular entre 0.4 y 0.5 segundos, y el menos habitual es un tiempo de ciclo también irregular entre 0.2 y 0.25 segundos). Por este motivo, hemos optado por la implementación de un Timer en Java.

- Debido a la mala gestión de los recursos por parte de *Pogamut*, en ocasiones el cálculo de rutas no se calcula correctamente, por lo que el comportamiento de dichos bots es erróneo e incluso se llegan a producir errores irrecuperables en su ejecución.
- Pogamut permite ejecutar varios bots en paralelo en la misma computadora mediante una sola instrucción. No obstante, una vez ejecutados dichos bots, deberemos esperar a que termine su ejecución para poder lanzar el siguiente "paquete" de bots. Esto impide la ejecución en paralelo de varios "paquetes" de bots, tanto en la misma computadora como en computadoras diferentes.

En cuanto a las simulaciones que hemos realizado, nos hemos dado cuenta la importancia, por un lado, del estudio analítico previo que nos permite definir una correcta función de adecuación y por otro, las técnicas que empleemos para analizar los resultados obtenidos. Los algoritmos genéticos han sido una pieza clave para resolver nuestros problemas, puesto que nos han permitido realizar una tarea de búsqueda y ajuste a soluciones óptimas. Modelos y recursos como las cadenas de Markov y la regresión logística nos han sido de gran utilidad para poder implementar nuestros experimentos correctamente y poder sacar conclusiones acertadas sobre nuestros resultados.

7.1. Trabajo futuro

Durante la realización de este proyecto se ha realizado un manual de Pogamut para programadores (ver Anexo A), por lo que los trabajos futuros podrán montar su sistema rápidamente y comenzar directamente a realizar sus simulaciones. También se podrá emplear las clases en Java creadas que implementan algoritmos genéticos, para ajustar diferentes implementaciones de los bots que se programen. Tras la realización de este proyecto se proponen a continuación algunas de las posibles líneas de trabajo futuras:

- Probar nuevas técnicas y metodologías de IA sobre la plataforma UT2004 + Pogamut
- Emplear más variables de entrada al sistema (percepción del bot) y su vez generar más variedad de salidas del bot (acciones) para implementar bots más reactivos.
- Trabajar con las nuevas actualizaciones de Pogamut, las cuales permiten obtener más y mejores datos, de esta manera se podría depurar el código obteniendo seguramente mejores resultados, especialmente en lo referente a movimientos y disparos del bot
- Contribuir al desarrollo de la plataforma Pogamut, de forma que ésta permita la paralelización de evoluciones de cara al aprendizaje evolutivo.
- Introducir nuevas hipótesis analíticas derivadas de las ya existentes. Nuevas funciones de ajuste, nuevos dilemas E-E, etc.

Glosario

- Bot: Agente autónomo virtual.
- E-E: exploration vs exploitation Dilemma.
- IA: Inteligencia artificial.
- Listener: Evento manejable por el programador, creado por el entorno Unreal.
- SISOSIG: Should I Stay Or Should I Go.
- NPC: Siglas del término inglés "Non-Playing Character". Utilizado como sinónimo de bot y agente autónomo.
- Juegos de Shooter: Juegos de carácter bélico en los que el objetivo del usuario es abrirse camino a través del juego disparando a todo jugador que se ponga a tiro.
- FPS: Siglas del término inglés "First Person Shooter". Genero de videojuegos de disparos en primera persona.
- TCP/IP: Modelo de descripción de protocolos de red. El modelo TCP/IP, describe un conjunto de guías generales de diseño e implementación de protocolos de red específicos para permitir que un ordenador pueda comunicarse en una red.
- UnrealScript: Lenguaje pensado exclusivamente para desarrollar contenido para juegos que usen el motor Unreal. Está basado en Java y C++. Es un lenguaje de programación orientado a objetos (OOP - Object Oriented Programming) lo que significa que está basado en los conceptos de clases y herencias. Es importante hacer notar que UnrealScript no tiene todas las características de un lenguaje de programación más completo.
- API: Interfaz de programación de aplicaciones (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Bibliografía

[J. Laird, 2000] Laird, J. E. It Knows What You're Going To Do: Adding Anticipation to a Quakebot (2000) 1.1, 1.2.2

[Gemrot et al., 2009] Gemrot, J. and Kadlec, R. and Bída, M. and Burkert, O. and Píbil, R. and Havlícek, J. and Zemcák, L. and Simlovic, J. and Vansa, R. and Stolba, M. and Plch, T. and Brom, C. Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. Agents for Games and Simulations: Lecture Notes in Computer Science, 2009, Volume 5920/2009, 1-15, DOI: 10.1007/978-3-642-11198-3 1. 2009. 2.1, 5.1 (document), 1.1

[Holland, 1992] John H. Holland, Adaptation in Natural and Artificial Systems, MIT Press, 1992. (document), 1.1, 2.2, 3.3.2

[Mind, 1950] Turing, A.M. Computing Machinery and Intelligence. Mind 49: 433-460. 1950. 1.3 1.2.1

[Thrun, 1992] S. B. Thrun, "Efficient exploration in reinforcement learning", Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University, 1992. 2.2

[Reynolds, 1987] Reynolds CW (1987). "Flocks, herds and schools: A distributed behavioral model". Computer Graphics 21 (4): 25–34. doi:10.1145/37401.37406. ISBN 0-89791-227-6.owards an ai behavior toolkit for games", AAAI Symposium on AI and Interactive Entertainment, 2001.

[Buro, 2005] M. Buro, "ORTS: A Hack-Free RTS Game Environment", Proceedings of the International Computers and Games Conference 2002, Edmonton, Canada.

[Buro, 2003] M. Buro,"Real-Time Strategy Games: A new AI Research Challenge", Proceedings of the International Joint Conference on AI 2003, Acapulco, Mexico.

[Goldberg, 1989] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA. 3.3.2

[T. Furtak, 2004] M. Buro and T. Furtak, "RTS Games and Real-Time AI Research", Proc. of the Behavior Representation in Modeling and Simulation Conference (BRIMS 2004), Arlington VA.

[Hoang el al, 2005] Hoang, H., Lee-Urban, S., and Muñoz-Avila, H. Hierarchical Plan Representations for Encoding Strategic Game AI . Proceedings of Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05). AAAI Press.

[Sukthankar, 2007] G. Sukthankar and K. Sycara." Policy Recognition for Multi-Player Tactical Scenarios". Proc. of Int. Conf. on Autonomous Agents and Multi-agent Systems (AAMAS), 2007.

[Brooks, 1987] R. A. Brooks, "Planning is just a way of avoiding figuring out what to do next", Technical report, MIT Artificial Intelligence Laboratory. 1987.

[Arrabales, 2009] R. Arrabales, A. Ledezma and A. Sanchis. Establishing a roadmap and metrics for conscious machines development. Proceedings of the 8th IEEE International Conference on Cognitive Informatics, to be published 2009.

[Lemke, 2008] A. Lemke and L. Zilmer-Perdersen, "Virtual Evacuation Training using Interactive Cognitive Agents," Master's Thesis. Technical University of Denmark, 2008.

[van Lent el al, 1999] van Lent, Laird, J. E., Buckman, J., Hartford, J., Houchard, S., Steinkraus, K., and Tedrake, R. Intelligent Agents in Computer Games, Proceedings of the National Conference on Artificial Intelligence, July 1999, Orlando, FL, pp. 929-930.

[Ferber, 1999] Ferber. J. Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence. Addison Wesley, London, 1999. 2.1

[Wesson et al., 1988] Wesson R.: et all "Network Structures for Distributed Situation Assessment". Readings in Distributed Artificial Intelligence, Ed. Alan H. Bond and Les Gasser, Morgan Kaufmann 1988. 2.1

[Sang Hoon et al., 2009] Sang Hoon Kang, Maolin Jin, and Pyung Hun Chang. "A Solution to the Accuracy/Robustness Dilemma in Impedance Control" 2009. 2.2

[Diego F. et al, 2009] Diogo F. De Oliveira, Anne M. P. Canuto, and Marcilio C. P. De Souto. 2009. The diversity/accuracy dilemma: an empirical analysis in the context of heterogeneous ensembles. In Proceedings of the Eleventh conference on Congress on Evolutionary Computation (CEC'09). IEEE Press, Piscataway, NJ, USA, 939-946. 2.2

[Kramer, 2001] D. Kramer and R. McLaughlin. The behavioural ecology of intermittent locomotion. American Zoologist, 41:137–153, 2001.

[O'Brien et al, 1989] W. J. O'Brien, B. I. Evans, and H. I. Browman. Flexible search tactics and efficient foraging in saltatory searching animals. Oecologia, 80(1):100–110, 1989.

[Sonerud, 1992] A. Sonerud. Search tactics of a pause-travel predator: adaptive adjustments of perching times and move distances by hawk owls (surnia ulula). Behavioral Ecology and Sociobiology, 30(3): 207–217, 1992

[Langton, 1989] C. G. Langton, "Artificial Life," Artificial Life, vol. 73, no. 1-2, p. xxix + 655, 1989. 2.1.1

[Terzopoulos, 1999] D. Terzopoulos, "Artificial life for computer graphics," Communications of the ACM, vol. 42, no. 8, pp. 32-42, Aug. 1999. 2.1.1

[Bajec et al, 2007] I. L. Bajec, N. Zimic, and M. Mraz, "The computational beauty of flocking: boids revisited," Mathematical and Computer Modelling of Dynamical Systems, vol. 13, no. 4, pp. 331-347, Aug. 2007. 2.1.1

[Berg, 1993] Berg, BC. Random Walks in Biology. Princeton University Press. 1993 2.1.1

[Russel et al, 2003]] R. A. Russell, A. Bab-Hadiashar, R. L. Shepherd, and G. W. Gordon, "A comparison of reactive robot chemotaxis algorithms," Robot. Auton. Syst., vol. 45, pp. 83–97, 2003. 2.1.1

[Grasso et al, 2000] Grasso, F.W. , T.R. Consi, D.C. Mountain and J. Atema (2000) Chemo-Orientation in Turbulence with a Biomimetic Robot Lobster. Journal of Robotics and Autonomous Systems. 30:115-131. 2.1.1, 3.1

[Barlow, 1969] H.B. Barlow, Sensory Communication. MIT Press, 1969. 2.1.1

[Hamza, 2006] Hamza, MH, Robotics and Applications. Salzburgo ACTA Press, 2006. 2.1.1

[Sutton et al, 1998]] Sutton, R.S. & Barto, A.G.. Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning). MIT Press, Cambridge, MA (1998). 2.1.2

[Vergassola et al, 2008] Massimo Vergassola, Emmanuel Villermaux & Boris I. Shraiman, "'Infotaxis' as a strategy for searching without gradients", in Nature, volume 445, pages 406–409 (2007 January 25); 2008 (document), 1.1, 3, 3.1, 3.1

[Spivey, 2007] Spivey, J. (2007). The continuity of mind. New York: Oxford University Press.

[Busemeyer et al, 2006] Busemeyer,T., Ryan K. Jessup, Joseph G. Johnson, James T. Townsend (2006). Building bridges between neural models and complex decision making behaviour. Neural Netw. 2006, 19(8):1047-58.

[von Mammen et al, 2009] von Mammen, S., Jacob, C. (2009). Swarming for Games: Immersion in Complex Systems. In Applications of Evolutionary Computing, EvoWorkshops 2009. Springer. 3.2

[Spalazzi, 2001] Luca Spalazzi, A Survey on Case-Based Planning. Articial Intelligence Review 16: 3–36, 2001 (document), 2.2, 2.3

[Oviedo, 2005] Oviedo, J. M. (2005). *Programación Dinámica. La Ecuación de Bellman y el Teorema de la Envolvente*. Universidad Nacional de Córdoba - Argentina