Introspección en sistemas de planificación basada en casos aplicados a juegos de estrategia

Trabajo Fin de Máster en Sistemas Inteligentes

Fernando García González

Máster en Investigación en Informática, Facultad de Informática, Universidad Complutense de Madrid

Septiembre 2011

Documento maquetado con TEXIS v.1.0+.

Este documento está preparado para ser imprimido a doble cara.

Introspección en sistemas de planificación basada en casos aplicados a juegos de estrategia

Informe técnico del departamento
Ingeniería del Software e Inteligencia Artificial
09/2011

Dirigida por el Doctor
Pedro Antonio González Calero y Antonio A. Sanchez Ruiz-Granados

Máster en Investigación en Informática, Facultad de Informática, Universidad Complutense de Madrid

Septiembre 2011

A mi familia y amigos
que hicieron esto posible.

A todos los lectores
espero que este trabajo os abra una nueva perspectiva.

Agradecimientos

Cuando bebas agua, recuerda
agradecerselo a la fuente

Proverbio chino

Hace mucho tiempo nunca me hubiese planteado hacer una tesis de máster, el esfuerzo, el tiempo y mis propias capacidades me lo impedían, pero poco a poco y gracias al esfuerzo tanto mío como de la gente que me rodeaba, mi aprecio por la informática y por la inteligencia artificial me ha llevado hasta el sitio donde estoy. Por ello quiero agradecer a toda la gente que ha estado a mi lado la ayuda prestada.

En primer lugar a mi familia, porque sin ellos nada de esto hubiese sido posible, ellos me han ayudado a seguir adelante y me han guiado por el camino que me ha conducido hasta aquí queriéndome y apoyándome en los momentos más difíciles de este viaje.

También a mis amigos, gente que tanto en la informática como fuera de ella me han apreciado y ayudado en este empeño, todos quisimos estar haciendo una tesis de este estilo y yo he sido el primero, pero espero que pronto me alcancéis.

Por último, y no por ello menos importante, a todos aquellos profesores y alumnos que anduvieron el camino conmigo, porque ellos me han hecho este viaje sea mucho más fácil de lo que esperaba que fuese. Gracias en especial a Pedro Antonio González Calero, quien me ayudo durante todo el desarrollo de la tesis, a Antonio Sanchez Ruiz-Granados, que me hizo de enlace con Pedro y me dio valiosos consejos sobre cómo desarrollar el proyecto, y cómo no a Santi Ontañón, creador de Darmok2, gracias por tu infinita paciencia y apoyo en este camino.

Abstract

In this project the experiments related to artificial inteligence, specifically the CBR (Case-Based Reasoning) systems applied to Real-time strategy, are explained.

The case based reasoning systems applied to this kind of games need to manage vast ammounts of information from the domain and make multiple decisions in real time on dynamic enviroments, appart from that, the complexity of this systems make it hard to detect and correct problems in the OLCBP (On line case-based planning) cicle.

With the idea of helping the expert we proppose to use introspection techniques that will analyze the behavior of this system during its execution. This system will take care of compiling, analysing and sampling the data on the possible problems concerning the enviroment.

The final objective of this work is to show in the most convincing manner an instroppection of the CBR systems and the implementation of a tool that will assisnt in the decision making of these enviroments.

Key words: Introspection, Darmok, CBR Systems, Adaptation CBR systems, CBR systems Similarity.

Resumen

El conocimiento es la mejor inversión
que se puede hacer

Abraham Lincoln

En esta tesis se explican los experimentos realizados en el ámbito de la inteligencia artificial y más concretamente en el de los sistemas CBR (Case-Based Reasoning) aplicados a juegos de estrategia en tiempo real.

Los sistemas de estrategia basada en casos, aplicados a este tipo de juegos necesitan gestionar mucho conocimiento dependiente del dominio y tomar decisiones en tiempo real sobre entornos muy dinámicos, además, la complejidad de estos sistemas hace que no siempre sea fácil detectar y corregir problemas en el ciclo OLCBP.

Con objeto de ayudar al experto proponemos usar técnicas de introspección que analicen el comportamiento del sistema durante su ejecución. Estos sistemas se encargan de la Recopilación de datos, el analisis de los mismos y el muestreo de posibles problemas en el entorno.

El objetivo final de este trabajo es exponer de la forma más convincente posible una introspección a los sistemas CBR e implementar una herramienta que ayude en estos entornos a la toma de decisiones.

Palabras clave: Introspección, Darmok, Sistemas CBR, Adaptación en sistemas CBR, Similitud en sistemas CBR.

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

Encontrar un buen principio es la mitad de todo trabajo

Pitágoras

1.1. Introducción

En este apartado se describen los objetivos de este proyecto, la estructura de la memoria y la metodología usada. Con ello se pretende que el lector se haga una idea de la manera en la que va a estar estructurada la memoria y pueda seguir de una forma mas simple el desarrollo de la misma.

1.1.1. Objetivos de la tesis

En primer lugar, la tesis tiene como objetivo ayudar a un experto a detectar y corregir errores, con objeto de esto, usamos técnicas de introspección que permiten visualizar de una forma clara lo que en cada momento está haciendo un sistema de inteligencia artificial, así como el porque lo está haciendo. Con esto el experto debe ser capaz de mejorar el sistema por medio de la utilización de la información que le facilitamos.

1.1.2. Estructura de la memoria

Esta memoria consiste en una descripción del estado del arte actual, un estudio de la introspección aplicada a nuestro entorno y una explicación de los experimentos.

El primer capítulo es una introducción a los sistemas CBR, su uso como herramientas de planificación, y un análisis de sus ventajas y problemas para esta labor. Esta sección también introduce una competición basada en el juego StarCraft para probar la aplicación de técnicas de IA, como el CBR, a problema de planificación.

Tras esta primera aproximación se presenta Darmok2, un sistema CBR que se utilizará en esta tesis para hacer los casos prácticos y demostrar las conclusiones a las que se pretende llegar. Su uso ha sido necesario para comprobar las distintas teorias planteadas así como para descubrir los mayores problemas de esta tecnología.

Para continuar en el capítulo tres se explican los problemas de adaptación en los sistemas CBR, expresando sus fallos y dónde es más probable encontrar huecos en el sistema, así mismo también se explican las teorias que existen y las distintas aproximaciones al mismo problema que conllevan estos huecos.

A continuación y ya entrando más en detalles tenemos el capítulo cuatro, un capítulo dedicado a la introspección en sistemas CBR desde un punto de vista práctico. En este apartado se presentarán los distintos valores encontrados el los apartados anteriores que serán estudiados en los experimentos así como las comprobaciones que se harán en la tesis, se plantearán distintas posibilidades y se verá que es más importante a la hora de hacer los experimentos intentando encontrar un compromiso entre usabilidad y facilidad a la hora de generar pruebas. También se explicarán los procesos que se llevan a cabo para sacar esta información y cómo su obtención puede ayudar a mejorar los sistemas usados actualmente, para por ultimo en el capitulo cinco y seis explicar los experimentos que se van a realizar y los resultados que han dado, en estos capitulos se verá más en detalle el trabajo realizado y se hará un poco más de énfasis en los resultados viendo con varias tablas e imágenes cuáles son los experimentos y las conclusiones que se sacan de esta información, sin olvidar en ningun momento el objetivo del trabajo y el trabajo futuro que se espera realizar en el mismo.

En los apéndices se pretende dar una introducción a cualquier persona que quiera seguir el camino que marca esta tesis y continuar con el trabajo realizado en ella. Para ello se explican en detalle los procesos de instalación y carga de los distintos componentes, se ve como comprobar si todo está bien montado y cómo se sigue la arquitectura de la que se ha hablado en los apartados anteriores.

También se explica como desarrollar inteligencias prácticas y ver a un nivel muy básico una batalla entre tu inteligencia y el sistema del juego, todo esto desde un punto de vista superficial, pero que da una introducción bastante buena a como iniciarse en el mundo de darmok2, de los sistemas CBR y de la planificación basada en casos para juegos de estrategia en tiempo real.

1.1.3. Metodología usada.

La metodología usada se centra en la recuperación por medio de la observación de los posibles problemas del sistema CBR (fallos en la recuperación y adaptación de casos), con ello y usando técnicas de introspección se puede

conseguir un mecanismo que permita a un experto la interacción de forma amigable con el sistema de inteligencia artificial, en nuestro caso con Darmok2. Para demostrar esta realidad, se utiliza StarCraft, un juego de estrategía en tiempo real en el que las tropas de varios ejercitos luchan entre si con objeto de exterminar a las de sus rivales. En esta tesis se utiliza únicamente una de las tres razas de StarCraft (Terran) con objeto de mantener un entorno controlado en el que hacer las pruebas.

Una vez generados los casos de prueba, los escenarios que se van a utilizar y la inteligencia artificial que se va a usar en cada ejemplo, se procede a la recuperación de la información obtenida y a su visualización con objeto de aplicar las técnicas de introspección y a generar una serie de informes que faciliten al experto la toma de decisiones en cuanto a la carga y diversidad de los casos de uso que se utilizan para poblar al sistema.

Capítulo 2

Estado del arte

Si buscas una buena solución y no la encuentras, consulta al tiempo, puesto que el tiempo es la máxima sabiduría.

Tales de Mileto

2.1. Introducción

En este apartado se van a estudiar los sistemas de razonamiento basado en casos y más concretamente Darmok 2 (19), de igual manera se mostrará una API para crear inteligencia artificial sobre juegos de estrategia en tiempo real como StarCraft, con idea de posteriormente aprovechar el conocimiento adquirido para crear una serie de técnicas de introspección que permitan la mejora de este sistema a lo largo de la tesis del máster.

En los últimos años la inteligencia artificial y el aprendizaje automático han ido evolucionando de forma muy significativa y los primeros algoritmos que se plantearon han sido reemplazados por teorías cada vez mas sólidas y firmes, empero, poco a poco estos algoritmos han ido usandose para resolver problemas más complejos hasta llegar a la situación actual en que la complejidad de los algoritmos comienza a dar problemas reales en velocidad y espacio de memoria requerido. Debido a esto se han planteado varias soluciones, el uso de aproximación basada en casos tal y como se explica en (7) y en (1) soluciona algunos de estos inconvenientes, por lo que a lo largo del artículo iremos viendo cómo funciona Darmok2, los sistemas de OLCBR (planificación basada en casos aplicada a sistemas en linea) de los que hablaremos posteriormente, el razonamiento basado en casos, la planificación basada en casos, y en general las tecnologías que se van a implementar a lo largo de la tesis, así como qué ventajas y que inconvenientes tienen y el modo de solucionar estos últimos. También para terminar explicaremos las conclusiones a las que hemos llegado y el trabajo que se pretende hacer en el futuro.

2.2. Razonamiento basado en casos

El razonamiento basado en casos (1) es un proceso que se encarga de solucionar problemas por medio de la búsqueda de situaciones anteriores similares a la actual y aplicar el mismo procedimiento para solucionar el problema dado. Los sistemas de razonamiento basado en casos (CBR) cuentan con una base de casos antiguos con objeto de reutilizarlos en las nuevas situaciones que se presenten. Para ello esta metodología cuenta con las siguientes cuatro fases:

- Fase de recuperación: Esta fase se encarga de la recogida de datos de la base de casos y la búsqueda de la situación más parecida posible al planteamiento actual de forma que cueste menos su transformación y posterior aplicación.
- Fase de reutilización: Esta fase es la que acomete la transformación del caso recibido desde la fase anterior adaptándolo a la situación actual.
- Fase de revisión: Esta fase se hace cargo de la tarea de evaluar la calidad de la solución propuesta y de su modificación en caso de que el resultado no haya sido todo lo satisfactorio que se hubiese deseado, de forma que la siguiente vez que se encuentre un problema del mismo estilo el resultado sea el deseado.
- Fase de retención: Esta fase no se lleva siempre a cabo y se encarga de almacenar el problema y su solución como un nuevo caso que podrá ser utilizado para resolver planteamientos similares en el futuro y la almacena para que sirva de caso base para otras búsquedas que involucren una situación similar a la que se ha visto actualmente.

El ciclo CBR se puede ver en la figura 2.1.

2.2.1. Planificación Basado en Casos

El método CBR ha encontrado una de sus mayores aplicaciones en la resolución de problemas de planificación, es decir, problemas en los que es importante tener una planificación de acciones a realizar tales cómo planteamientos para recetas de cocina o juegos de mesa. En este tipo de problemas, las acciones secuenciales se tratan como distintos planes en los que los pasos llevados a cabo con anterioridad se convierten en un prerrequisito para la acción que se lleva a cabo, tal y como se explica en (8).

Esta planificación por medio de CBR recibe el nombre de CBP (planificación basada en casos) y cuenta con un gran número de estudios y algoritmos que la han ido mejorando a medida que ha pasado el tiempo.

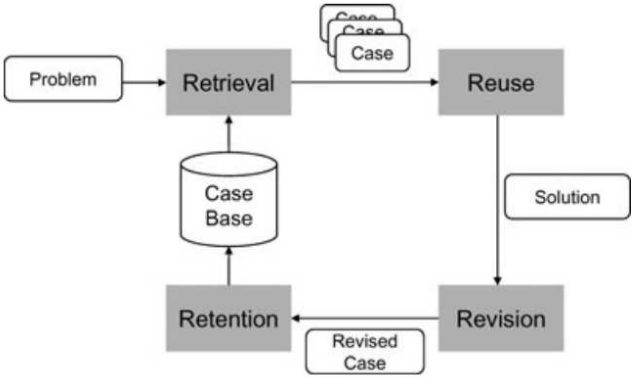

Figura 2.1: Ciclo CBR

Podemos distinguir dos tipos de sistemas CBP en función de si los planes tienen una estructura lineal o jerárquica, o de forma mas común, aquellos que tienen un plan completo que representa todo el objetivo del sistema, y aquellos que usan planes parciales para generar un plan completo (estos planes deben seguir una estructura en cascada) en la que cada plan principal se divida en una serie de planes secundarios.

2.2.2. On-Line CBP y juegos de estrategia en tiempo real

Los juegos de estrategia en tiempo real se han convertido en uno de los generos más expandidos de la actualidad tal y como se ve en (22). Existen ahora mismo muchas competiciones oficiales e incluso una liga de StarCraft (uno de lo juegos mas famosos dentro de este género). Por lo que este género se convierte en un buen dominio de pruebas para los sistemas CBP.

Algunos de los principales retos a los que se enfrentan estos sistemas son:

- Puede llegar a tener un repositorio de casos tremendamente amplio y complejo
- El tiempo de respuesta no es infinito, es decir, hay que tomar una decisión en un tiempo corto aunque no sea la mejor posible.
- Los juegos de estrategia son no deterministas, lo que implica que las acciones a veces pueden fallar y se tiene que tener en cuenta esta casuística.

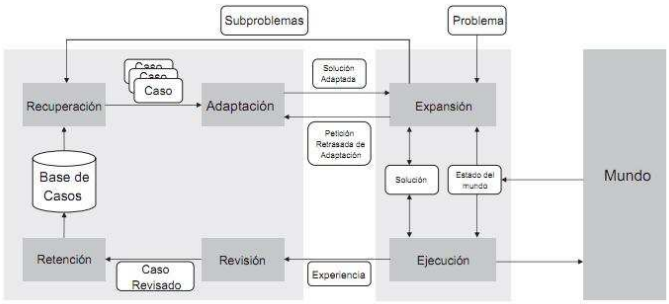

Figura 2.2: Ciclo CBR expandido

- El jugador no tiene toda la información, el mundo va cambiando y es necesario que el sistema adapte sus planes dinámicamente.
- En este tipo de juegos se tiene un adversario, lo que conlleva que mientras el programa piensa su estrategia dicho adversario no se queda parado, sino que sigue haciendo cosas y mejorando su situación, lo que a su vez provoca cambios en el mundo.
- El modelado del dominio no es trivial, por ejemplo, existe bastante dificultad en la elección de postcondiciones para las acciones.

2.3. Darmok2

Darmok2 es un sistema de planificación basada en casos especializado en OLCBP y que se utiliza sobre todo para juegos de estrategia en tiempo real, aunque también es útil en cualquier tipo de juego de competición entre varios jugadores. Este juego implementa un sistema de aprendizaje mediante ejemplos, basado en un usuario experto que se dedica a jugar y con ello va generando una serie de trazas que sirven para que el sistema aprenda en base a las demostraciones prácticas del jugador.

Darmok2 usa un plan jerárquico que se divide en distintos subobjetivos de manera que el planteamiento inicial de ganar la partida se va dividiendo en otros objetivos más sencillos, y poco a poco estos objetivos se van descomponiendo sucesivamente, hasta llegar a una situación en la que todos los objetivos sean indivisibles, de forma similar a las redes jerárquicas de tareas.

2.3.1. Ciclo CBR expandido

Las redes jerárquicas de tareas siguen un plan denominado HTN tal y como se explica en (14), dicho plan jerárquico sigue un concepto similar a los CBR en línea (OLCBR) la diferencia principal consiste en que los sistemas OLCBR asumen que el problema se puede ser resuelto con una única busqueda en su base de casos (si los objetivos no se subdividen ni se hace una planificación lo lógico es que una consulta a la base de casos de una respuesta y el ciclo se termine), Darmok2 introduce dos nuevos pasos al ciclo

Las dos nuevas fases que se implementan quedan descritas a continuación:

- Expansión: Esta fase se encarga de ampliar el número de planes por medio del desarrollo de los árboles de objetivos de la forma siguiente, primero coge el objetivo que se le da y lo desensambla mirando si tiene algún subobjetivo que se deba cumplir primero, y caso de que lo encuentre lo recupera y lo manda a la fase de recuperación (la primera del ciclo) para su resolución previa a continuar el plan actual.

- Ejecución: La ejecución es la encargada de llevar a cabo el plan y modificar de forma apropiada el estado del mismo, de forma que si algunos de los subplanes fracasa este paso se encargara de su notificación con objeto de que se plantee una forma alternativa de solucionar el mismo.

2.3.2. Planes de Darmok2

Darmok2 como ya hemos comentado con anterioridad se basa en planes aprendidos a partir de las trazas generadas automáticamente en las partidas reales tal uy como se ve en la figura 2.3, y que se plantean de forma que una serie de prerrequisitos satisfacen un objetivo, por ejemplo, *(si tienes X unidades del material Y construye Z)*. Pero claro, para que eso funcione es necesario que se hagan varias cosas, primero es necesario que las trazas que se consiguen por medio del usuario final se conviertan en planes que se organicen en la forma que se ha comentado y luego es necesario recuperar esos planes, adaptarlos y ejecutarlos. En este apartado veremos esos dos grandes apartados así como el software necesario para llevarlo a cabo.

2.3.2.1. Representación de los planes en Darmok2

El sistema básico de representación de un plan en Darmok2 es el Snippet, que grosso modo es la representación de un conjunto de tareas a realizar y que consta de las siguientes partes:

- Precondiciones: la lista de precondiciones indica las tareas u objetos que hay que tener terminadas para que el Snippet pueda ser realizado,

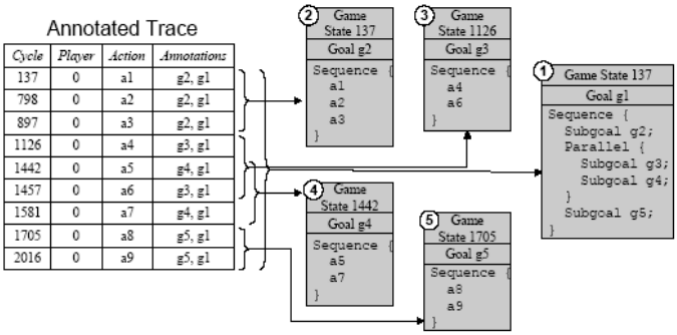

Figura 2.3: Extracción de casos en Darmok a partir de trazas

los materiales necesarios para una construcción o la unidad apropiada para la realización de una tarea

- Condiciones de vida: la lista de condiciones que tienen que mantenerse ciertas durante toda la aplicación del plan para que este se considere un éxito, es decir, si alguna de estas condiciones deja de ser cierta el plan automáticamente ha fallado.
- Condiciones de éxito: Condiciones que indican que el plan ha tenido éxito.
- Plan: El conjunto de acciones a realizar.

Los planes realizados se almacenan en una red de petri jerárquica (1), que permite descomponer un plan global en subplanes.

Por otro lado para la recepción de los planes por medio del usuario experto es necesario dar al sistema información sobre el dominio del juego, que se consigue a través de un fichero en lenguaje MakeMEPlayME un framework con capacidad para conectar sistemas de AI con juegos de estrategia en tiempo real que divide el dominio en acciones básicas (moverse, atacar...), sensores (que permiten ver el estado del mundo) y objetivos (puestos además en una jerarquía)

2.3.2.2. Utilización de los planes en Darmok2

Los planes en Darmok2 son almacenados con un formato especial que permite de una forma sencilla su recuperación y su evaluación. Para ello, se

genera una red de petri que consta de casos que siguen el formato c = (p, e) donde p es el Snippet y e representa el episodio de la forma e= (G, S, O) siendo G el objetivo perseguido para ese episodio, S el estado del mundo en el momento en que se llevo a cabo y O un valor entre 0 y 1 que marca la medida del éxito del plan p por medio del episodio e. La primera fase como ya hemos visto si usamos planes de Darmok2 es la recuperación de planes y esto se hace viendo qué episodio cumple de una forma mejor el objetivo que se tiene en mente, para ello se usa la formula siguiente:

ER (e; S; G) = α GS (e:G;G) + (1- α)SS(e:S; S)

Donde GS es la medida, evaluada entre 0 y 1, de la distancia entre objetivos, SS es la medida, también entre 0 y 1, entre los estados del mundo y α es un numero entero que indica lo importante que es cada una de esas medidas de distancia. Por último se obtienen los episodios más relevantes según la fórmula vista y se toma el valor que mejor cumpla el objetivo planteado.

2.4. Introspección en sistemas CBR

En este apartado explicaremos el uso de la introspección en el desarrollo de los sistemas CBR, para ello primero explicaremos en qué consiste exactamente el uso de la introspección como fuente de mejora en un sistema y luego adentrándonos más en los sistemas CBR veremos en qué se puede usar para este caso concreto. Una vez entrados en materia pondremos varios ejemplos y mostraremos algunas ideas que se barajan actualmente en este campo con objeto de dejar la información lo más simple posible.

2.4.1. Introspección para el desarrollo de tecnologías

El uso de la introspección para el desarrollo de teorías ha sido siempre una de las mejores fuentes a la hora de obtener resultados, la introspección que se define como el conocimiento que el sujeto tiene de sus propios estados mentales. Esto en el caso de los procesos implica tres cosas.

- Se debe tener el objeto a estudiar
- De este objeto se tienen que sacar una serie de informaciones que lo definan dentro de un ámbito concreto.
- Estas informaciones tienen que ser conseguidas de primera mano y por medio de experimentos directos.

Una vez cumplidos los tres pasos anteriores podremos decir que tenemos una introspección de un trabajo, pero con eso no vale, la introspección también necesita un objetivo, es decir, necesita una finalidad que le de forma, una idea a demostrar ya sea por lo válido de la misma o por su error. Y una

vez que tenga esto la introspección debe dar respuesta a la pregunta que se formula y con objeto de la cual se ha hecho el entorno.

2.4.2. Uso de la introspección en sistemas CBR

Los sistemas CBR han usado la introspección prácticamente desde el momento en que fueron concebidos en 1985 ya que la calidad de los sistemas CBR dependen de la experiencia y la calidad de los casos que lo forman.

Ademas del razonamiento introspectivo es necesario en caso de los sistemas CBR, satisfacer los objetivos más claros del sistema, es decir, encontrar qué fallos tiene el entorno y que casos son los apropiados y cuales sobran en la búsqueda de una respuesta válida, para con esta información ir mejorando sus algoritmos y utilidad. En estos sistemas además hay una importante brecha que es difícil resolver, y que se da en el hecho de que una persona que es un verdadero experto en una materia es varios órdenes de magnitud mejor que un número alto de expertos que sean casi tan buenos como él, lo que lleva a una situación incómoda ya que no se puede dividir el trabajo y normalmente una persona que es verdaderamente experta en una materia es muy difícil de sacar de su entorno para que provea un sistema informático de la información que necesita para trabajar, e incluso en el caso de que se pueda sacar a este experto de su entorno muchas veces no será capaz de dar razonamientos verdaderamente lógicos de cómo se desarrolla una secuencia de acciones o un determinado tema de importancia que sea necesario cargar en el sistema CBR. Esto en los sistemas CBR al igual que en muchos otros aspectos de la inteligencia artificial, es el cuello de botella de una aplicación de este estilo.

2.4.3. El uso de la introspección para búsquedas en memoria en sistemas CBR

En esta parte se van a tratar algunos de los problemas más importantes que puede resolver el uso de la introspección sobre los sistemas CBR, y más concretamente en el ámbito de la búsqueda en memoria.

El problema que se plantea aquí es fácil de entender, un sistema CBR busca en un espacio de memoria los casos que se parecen a un objetivo, tal y como se vio en la sección anterior sobre estos sistemas. Sin embargo, cada vez este tipo de tecnología ambiciona más y pretende cubrir problemas más complicados. Por ello hay que mejorar todo lo posible el sistema y para ello una de las maneras es por medio de la introspección.

La adquisición de conocimiento si nos centramos en el modo en el que aprenden los sistemas CBR sobre un espacio de dominio es un tema que la introspección puede tratar por medio de la generación de una nueva forma de búsqueda en memoria basada en planes, así como la necesidad de adaptar la información de la memoria en el proceso de extracción, y con ello crear

algoritmos mejores y evitar el problema de la sobrecarga de casos en la aplicación.

Algunos de los puntos donde esto se puede ver son los siguientes:

- Generación de objetivos sobre un dominio de conocimiento.
- Razonamiento introspectivo de la forma en la que la búsqueda en memoria satisface estos objetivos.
- Mejora de las búsquedas en memoria por medio de estrategias

2.4.4. Adquisición de la experiencia por medio del razonamiento introspectivo

Es importante también tener en cuenta el número de casos que tiene la base de datos y otras situaciones que pueden empeorar el tiempo de respuesta o incluso el rendimiento del sistema.

Para evitar esto la introspección por medio de la intervención de un experto es muy importante, y como ya se ha comentado antes para ello es necesario facilitar al experto el muestreo del funcionamiento de los casos así como la capacidad de intervención directa de su parte con el sistema. Como resumen podemos decir que esta parte tiene que tener en cuenta los siguientes ítems.

- Usar el proceso CBR por sí mismo para encontrar fallos intrínsecos al mismo.
- Identificar los procesos y fallos ocultos que puede tener el sistema.
- Muestreo bidireccional entre un experto y la herramienta de la forma más simple posible para descubrir problemas en la base de casos.

En esta tesis se demuestra que por medio de la introspección, estos errores se pueden tratar, mejorando de forma considerable los sistemas CBR, aumentando la probabilidad de éxito en la recuperación de casos.

2.5. Competición de StarCraft

Antes de hablar de la competición de StarCraft es importante hacer un pequeño resumen del juego para aquellos que no lo conozcan.

Este juego se basa en tres razas que luchan en un escenario, su mecanica es puramente estratégica, y cada ejercito, manejado por un jugador usa los recursos del entorno para crear nuevas unidades, nuevos edificios y nuevas tecnologías con las que vencer a su rival, para más información se puede ver el siguiente enlace: (5).

La competencia de StarCraft AIIDE en 2010 se organizo como parte del programa de la conferencia por medio de la Expressive Intelligence Studio at UC Santa Cruz. El concurso permitió a los investigadores poder evaluar sus sistemas de IA en un entorno sólido comercial de estrategia en tiempo real. El concurso se llevó a cabo en las semanas previas a la conferencia ya que las batallas finales se debían mostrar en vivo en la misma. Además, se llevaron a cabo encuentros de exhibición entre jugadores humanos y aquellos bots de mejor rendimiento. Los bots de StarCraft se pueden desarrollar utilizando la API de Bloodwar, que proporciona la conexión con StarCraft y además permite el desarrollo de programas o módulos con AI para StarCraft. Una interfaz C++ permite a los desarrolladores poder consultar el estado actual, las órdenes de juego y dirigir a las unidades.

En la competición se hacen uso de técnicas avanzadas de inteligencia artificial. Algunas de estas técnicas son:

- Planificación
- Minería de datos
- Aprendizaje Automático
- Razonamiento Basado en Casos

A continuación vamos a explicar varias cosas, la primera la competición de StarCraft, comentando como se decide que bots son mejores y quien gana cada encuentro, del mismo modo se explicarán las fases que tiene la competición y los distintos torneos en los que se puede competir durante la misma. La segunda por otro lado, será la técnología que se permite y los modulos que se utilizan para interactuar con el juego durante esta competición.

2.5.1. Torneos de StarCraft (Dentro de una competición)

La competición se divide en cuatro torneos, en todos los torneos se utiliza un soporte de doble eliminación, basado en las mejores 5 partidas. Se insertarán en cada escenario los robots de los dos oponentes que competirán en la ronda y un robot adicional observador que hará de juez, la estructura se puede ver en 2.4.

2.5.1.1. Micro-managemet

El primer torneo se centra en micro-gestión de escenario con terreno plano. El torneo utiliza un soporte de doble eliminación. Los bots serán emparejados contra los bots de otros en encuentros 1 contra 1, donde un encuentro consiste en *El mejor de cinco*. Al comienzo de cada ronda del torneo, el orden en el mapa será seleccionado al azar. El objetivo de este primer torneo es destruir todas las fuerzas enemigas, además, la niebla en el

Figura 2.4: Sistema de competición para evaluar bots

escenario estará habilitada, lo que significa que las tropas solo verán aquello que tienen cerca y no todo el mapa.

2.5.1.2. Torneo: Small-Scale Combat

El segundo torneo se basa en el primer torneo pero con la adición de un terreno interesante en los mapas. En StarCraft, conservar un terreno alto ofrece dos ventajas: en primer lugar, las unidades inferiores no son capaces de ver a las unidades de tierra firme que se encuentren en una posición elevada y en segundo lugar, las unidades de terreno bajo al atacar unidades en terrenos altos tienen una tasa 70 por ciento de acierto. Por lo tanto, el posicionamiento es un aspecto importante del juego táctico.

El torneo utilizará un soporte de doble eliminación. Los bots de unos jugadores serán emparejados contra los bots de otros, por medio de partidos 1 contra 1, un partido lo gana el primer equipo que venza tres veces *el mejor de cinco* Los mapas serán seleccionados al azar al principio de cada ronda. El objetivo de este torneo es destruir todas las fuerzas enemigas, y se tendrá activada la niebla en el escenario, además, se contará como un empate y el partido se reanuda en caso de un estancamiento en la partida.

2.5.1.3. Tech-Limited Game

Este torneo se caracteriza por tener tecnología limitada, evalúa los robots en un entorno simplificado de StarCraft. El torneo exige razonamiento a nivel estratégico y táctico, pero omite la mayor parte de la complejidad de

la versión completa de StarCraft (encubrimiento, transporte y unidades de aire). En este torneo está restringido a una batalla de Protoss contra Protoss, además solo se pueden generar las siguientes construcciones:

- Nexus
- Pylon
- Gateway
- Assimilator
- Cybernetics Core
- Forge
- Shield Battery

Solo se pueden crear las siguientes unidades:

- Probe
- Zealot
- Dragoon

Solo se pueden tener las siguientes investigaciones:

- Singularity Charge (Dragoon range)
- Ground weapons (Level 1)
- Armor (Level 1)
- Shields (Level 1)

Los bots de unos jugadores serán emparejados contra los bots de otros, por medio de encuentros 1 contra 1, un encuentro consiste en *el mejor de cinco*. Los mapas serán seleccionados al azar al principio de cada ronda. El objetivo de este torneo es destruir todas las estructuras enemigas, y se tendrá desactivada la niebla en el escenario.

2.5.1.4. Complete StarCraft Game

En este torneo se evalúa los bots en el juego StarCraft en toda regla. Uno de los objetivos del torneo es la evaluación de los robots en un formato de competencia similar al que se enfrentan los jugadores profesionales tal y como se puede ver en (5).

2.5.2. BWApi

El nombre de BWApi viene de Blood War Application Programming Interface es un framework, de código libre y gratuito que está desarrollado en C++ y se puede conseguir de (18). La idea de BWApi es poder crear módulos AI para starcraft. Cuando se programa con BWApi se puede llegar a obtener información sobre los jugadores y además información sobre una unidad, por lo tanto uno tiene una puerta abierta para moldear la AI con micro o macro algoritmos. BWApi está compuesto de tres partes (AI module, AI client, AI module loader)

2.5.2.1. AI Module

Es un proyecto en el cual se nos enseña cómo construir una DLL, esta DLL es la forma estándar de crear agentes autónomos con Bwapi.

2.5.2.2. AI Client

Este modulo es un simple proyecto para poder desarrollar un programa cliente AI, una nueva manera experimental de realizar AI's con Bwapi.

2.5.2.3. ChaosLauncher

El ChaosLauncher es un programa que sirve como interfaz grafica para varios plugins de starcraft, el mismo soporta plugins usando el formato BWL4, además, posee una integración con el launcher, por lo que las DLL son directamente inyectadas al juego, para más información sobre el ChaosLauncher se puede mirar (13).

Capítulo 3

Problemas de la adaptación en sistemas CBR

No existen los problemas, solo los desafios

Enrique Barrios

3.1. Introducción

Como ya se ha comentado, la planificación basada en casos (CBP) es una especialización del razonamiento basado en casos que se centra en intentar planificar las situaciones de la forma más factible posible, en el documento (8), se puede encontrar una idea de cómo hacer esto, así como una serie de guías sobre como mezclar la planificación basada en casos y los juegos en tiempo real, a su vez, en (7) y en (15), se indica que es necesario perfeccionar el uso del CBR por medio de la inserción de otras tecnologías para cubrir sus fallas.

Los sistemas de CBP tienen una estructura que se centra en la reutilización de casos, que en juegos como el ajedrez o en programas donde hay una serie de movimientos definidos y que son deterministas, es decir, las acciones que se piden se llevan a cabo sin ningún error, tienen un conocimiento del dominio suficiente, pero en sistemas que no son deterministas y que pueden tener información contradictoria o no aplicable al entorno del caso objetivo, la situación cambia, y el ciclo CBR debe ser modificado hasta acabar de la forma en la que se muestra en la figura 3.1.

En la imagen se puede ver una serie de flechas que van desde la recepción de un problema hasta la elaboración y retención de la solución, de forma muy similar a como lo comentábamos en el capítulo anterior, en este diagrama existe un paso que marca una diferencia muy importante con respecto al diseño anterior y que es importante a la hora de ver los problemas que tienen

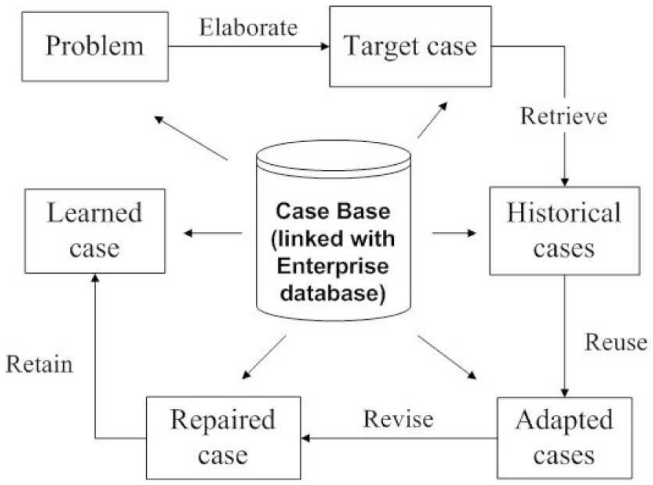

Figura 3.1: Ciclo CBR complejo

los sistemas CBR. El paso en concreto se describe a continuación:

- Elaborate: en este paso se hará la elaboración del caso objetivo que será resuelto, o intentado resolver.

Como se puede comprobar fácilmente, en este caso se tiene que tratar de simplificar en lo más posible (por cuestiones de facilidad de uso y de espacio en memoria) el estado del mundo que se representa e intentando no pasar nada por alto de lo que hace que el caso sea relevante para el sistema de aprendizaje y se resuelva de la forma en la que se ha llevado a cabo, es decir, hay que intentar en la medida de lo posible dar toda la información relevante al sistema siempre que se detalle un objetivo o bien se almacene el mismo en la base de casos con los pasos que llevaron a su solución o fracaso.

En los apartados que siguen detallaremos los distintos problemas a los que se debe enfrentar el sistema CBR y que serán tratados en los distintos experimentos realizados para intentar descubrir dónde y por qué falla el sistema y que deben contener los casos que es necesario introducir o retirar y cuyo objetivo será llevar a la mejora de la inteligencia artificial del mismo, creando así un sistema que cada vez reaccione mejor y más rápido al mundo cambiante de los juegos de estrategia en tiempo real.

3.2. Fallos de similitud

En este apartado nos centraremos en los fallos de similitud, estos fallos generan problemas en la actuación de la inteligencia, cosa que se puede ver

en el artículo (17)ya que impiden que se tomen medidas apropiadas para el contexto en el que nos movemos tal y como se puede ver en (10) y más en detalle en (9), se pueden generar en varias partes de la aplicación y el tratamiento es difícil ya que hay multitud de algoritmos que se pueden generar para calcular la similitud y para confirmar que el sistema se mueve dentro de los parámetros establecidos.

3.2.1. Sistemas de similitud

- Distancia Euclídea: Esta medida es la más usada. Definimos la distancia Euclídea entre dos puntos x e y como se ve en la siguiente fórmula, Donde x_i es la coordenada de x en la dimensión i y y_i es la coordenada de y en la dimensión i.

$$f(x) = \sqrt{sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.1}$$

- Distancia de Manhattan: La distancia de Manhattan entre dos puntos es la suma de los valores absolutos de las diferencias de sus componentes. De la forma en que se ve en la siguiente fórmula:

$$f(x) = sum_{i=1}^{n} |x_i - y_i| \tag{3.2}$$

- Alineamiento dinámico temporal: la función de similitud basada en el alineamiento dinámico (11) es sin duda la más robusta de las aquí tratadas. Y se basa en series temporales, indicando que si se quiere comparar la similitud entre dos series temporales (como puede ser el caso de los juegos de estrategia en tiempo real) la siguiente fórmula:

$$Q = q_1, q_2, ...q_i, ..., q_n; C = c_1, c_2, ...c_i, ..., c_n. \tag{3.3}$$

  Se construye una matriz W (n x n) en el que cada elemento (i,j) se corresponde con el alineamiento de q_i con c_j. Y definimos un camino de alineamiento como un conjunto de elementos contiguos de la matriz que definen una correspondencia entre Q y C. El elemento k-esimo de W se define como $w_k = (i, j)_k$ y por tanto:

$$W = w_1, w_2, ...w_i, ..., w_n \tag{3.4}$$

$$max(m, n) = K \geq m + n - 1 \tag{3.5}$$

- Por último el resultado lo dará el aplicar esta fórmula para cada variable. La semejanza entre 2 series multivariable será la media de las semejanzas para cada variable.

3.2.2. Principales causas de problemas de similitud

Los problemas de similitud pueden ser causados por tres causas principalmente.

- Fallos al generar el caso objetivo:

Este fallo se da cuando el sistema que genera el caso objetivo de búsqueda no recoge todo lo necesario para que el algoritmo pueda formarse una idea de lo que se necesita, eso conlleva una similitud baja en la mayor parte de los casos y en caso de ser una similitud aceptable el resultado no será, ni se acercará al esperado.

- Fallos en el algoritmo de similitud:

Este fallo es imposible de evitar de una forma completa, pero aunque no se pueda eliminar totalmente es primordial conseguir minimizar su efecto, se debe como ya hemos visto antes a la gran cantidad de algoritmos que existen para la toma de decisiones y a la obligación de alcanzar un compromiso serio entre fiabilidad del caso recuperado y el tiempo que se tarda en recuperarlo.

- Fallos en el sistema de almacenamiento de casos

Este fallo se da de forma similar al primero, es decir, de igual manera que el primero de los casos tratados generaba errores al convertir el estado del mundo a información válida para el sistema de decisión, este error se genera cuando se trata de almacenar un caso resuelto, ya sea positiva o negativamente, en nuestra base de casos, ya sea por tamaño que ocupa en memoria, o por falta de información relevante para la toma de decisiones.

3.3. Problemas de adaptación

Otro de los grandes problemas del razonamiento basado en casos se da en la adaptación de casos con objeto de reutilizarlos, en un juego como el que nos ocupa, es prácticamente imposible encontrar un caso que de verdad ya se haya dado, ya que las posiciones de recursos, tropas o edificios, el estado de las habilidades de las tropas, las mejoras que se han desarrollado, etc... son muchas y variadas, del mismo modo hay varias razas con un comportamiento distinto que multiplican el nivel de complejidad de la inteligencia.

Por ello merece una especial atención la adaptación de casos y con este fin se plantea este apartado como una introducción a las distintas tecnologías que se usan y a las múltiples variantes que hay ahora mismo en debate, algunas de dichas variantes pueden encontrarse en (16) y en (6).

3.3.1. Adaptación nula

Sea cual sea la aplicación elegida está claro que la adaptación más simple y la más rápida es esta, se basa en no hacer adaptación, además se considera una de las opciones viables ya que si uno se plantea un sistema simple en que las opciones están (más o menos) acotadas y el trabajo que tiene detrás el desarrollo de la inteligencia puede ser demasiado complejo para las ventajas que da, en estos casos, una adaptación tan solo conlleva un gasto de CPU y una complejidad excesiva del desarrollo de la inteligencia.

EJ1. Un sistema que acepte o rechace multas de tráfico (si una multa es muy parecida a otra que tenga una sanción casi con seguridad tendrá la misma sanción).

- Ventajas:
  - Algoritmo muy rápido a la hora de evaluar
  - Algoritmo fiable para respuestas binarias
- Desventajas
  - El algoritmo no es aplicable a casi ningún problema real
  - El algoritmo realmente no ayuda en nada a la adaptación.

3.3.2. Adaptación paramétrica

La adaptación paramétrica es probablemente la más comprendida y la más extendida, se basa en tratar de adaptar la situación modificando sus parámetros para que coincidan con el caso objetivo. Aunque esta solución es bastante simple de entender, no siempre es simple de implementar, ya que generar pares de parámetros (parametro origen, parametro destino) puede ser un trabajo difícil para el ordenador que no sabe si lo que está asociando es realmente suficientemente aproximado para no generar un error al hacer la adaptación, esto conlleva a que este tipo de soluciones puedan adaptar una solución para aproximarla al caso base, pero no son útiles para generar una nueva solución partiendo desde cero, es decir, se usa como interpolación y aunque es una buena forma de multiplicar el conjunto de casos base que se tienen, no siempre sirve para generar el numero mínimo de casos distintos que son necesarios para que un sistema funcione.

EJ. El estratego es un juego similar al ajedrez, en el que cada ficha pondera un valor (de uno en adelante), cuando dos fichas se encuentran gana la que tenga un valor mayor. Una adaptación parametríca haría que si el oponente tiene una ficha de tres, la jugada en la que se venció a la ficha del rival con una de valor cuatro se pueda sustituir por una jugada similar con una ficha de mayor valor.

- Ventajas:
  - Esta solución es más o menos sencilla de implementar
  - Es una forma muy potente de adaptar casos que ya se tienen si las reglas de adaptación están bien diseñadas.
  - En relación coste / beneficio se sale bastante bien parado en la mayoría de los sistemas
- Desventajas:
  - Este sistema no puede generar los primeros N casos bases
  - Es imposible aprender una casuística que no está ya diseñada, no aprende, solo explota el conocimiento. (Tengamos en cuenta que tomamos aprendizaje como adquisición de conocimiento de forma autónoma, por mucho que el sistema aumente su base de casos no puede aprender si no recibe nueva información con valor real que añadir)
- Programas que la usan:
  - HYPO (Rissland and Ashley, 1986; Ashley and Rissland, 1988) (3)
  - PERSUADER

3.3.3. Adaptación estructural

La adaptación estructural se encarga de generar una serie de reglas que se aplican directamente al caso extraído de la base de casos y que se encarga de generar una solución. Dentro de la adaptación estructural se encuentran muchos algoritmos de muy diversas formas, algunos de los más importantes se debaten a continuación.

3.3.3.1. Abstracción y reespecialización

Este método de adaptación estructural se considera uno de los más comunes, se plantea la solución de un problema por medio de la abstracción de las partes del mismo que no se consigue aplicar al problema actual, es decir, de una solución ya almacenada y que se ha elegido como válida para resolver el problema, si una parte en concreto no se puede aplicar, se busca una solución alternativa que complete la abstracción de la parte del problema que no fue válida, de esa forma se generan 'parientes'de la solución que se pueden utilizar. Ej. Darmok2 cuando intenta generar un edificio (barracas) elije un caso en el que esto se hiciese, supongamos ahora que el caso que recoge crea el edificio en un lugar en el que en la situación actual no se puede generar (terreno ocupado por algo), en ese caso, busca en la base de casos otro plan

que satisfaga la condición (crear edificio barracas), tras encontrarlo utiliza este caso para generar el edificio de forma satisfactoria.

- Ventajas:
  - Fácil de implementar
  - Da una adaptación realista y válida para muchos de los problemas que necesitan una adaptación mayor que la paramétrica.
- Desventajas:
  - Genera una búsqueda que puede llegar a ser recursiva e infinita.
  - El algoritmo puede ser muy lento hasta que encuentra una solución que satisface todas las precondiciones generadas.
- Programas que lo usan:
  - PLEXUS (Alterman, 1986) (2).

3.3.3.2. Adaptación basada en crítica

Esta técnica de adaptación estructural se basa en la utilización de críticas para la generación de soluciones correctas partiendo de soluciones casi correctas, nació en 1975 y fue desarrollada desde 1988. Se basa en la generación de un motor de adaptación por medio de la combinación de reglas en lugar de la recuperación de las mismas desde memoria. Una crítica es grosso modo, una combinación de características que pueda generar un fallo en el plan que se pretende poner en marcha. En las teorías de Sussman, basadas en un plan que debe cumplir varios objetivos, esto se consigue poniendo en conjunto todos los planes que cumplen los distintos objetivos independientes para intentar encontrar contradicciones en los mismos.

Por desgracia para este algoritmo, tiene las mismas restricciones que las adaptaciones parametrizadas, ya que no puede generar planes por sí mismo y tiene que ser bombardeado con planes independientes desde el exterior para que luego los use en sus adaptaciones.

EJ. El programa de recetas CHEF, obtiene soluciones básicas tomando las recetas anteriores y luego sustituyendo los ingredientes que tenían por los nuevos, a partir de ese momento, unas críticas sobre los ingredientes comprueban si los mismos tienen que sufrir alguna modificación, como por ejemplo, saltarse los pasos de cortado si son ingredientes más pequeños que los originales.

- Ventajas:
  - Se comprueban los planes usando varios ejemplos de forma simultánea.

  - La carga en memoria es menor ya que no usa la base de casos de la misma manera que el resto de algoritmos.

- Desventajas:
  - No es capaz de generar resultados desde cero y tiene que ser nutrida con una amplia y variada selección de reglas.
  - Hacen cambios locales en vez de reorganizaciones globales de forma muy similar a las adaptaciones paramétricas
- Programas que lo usan:
  - CHEF

3.3.4. Adaptación derivacional

La adaptación derivacional es con mucho, más compleja que la estructural, se basa en el almacenamiento conjunto de los casos y de los planes que dieron lugar a dichos casos, y en lugar de funcionar como la estructural que modifica la propia disposición, sino que cuando se recibe un objetivo nuevo el sistema aplica de nuevo el plan antiguo con los parámetros de entrada nuevos, para confirmar que el resultado de aplicar dicho plan será el esperado, si por el contrario los nuevos parámetros afectan a las decisiones del plan antiguo, dichas decisiones se reevaluarán con objeto de descubrir cambios con los valores actuales de entrada. Es decir, en resumen el código no se adapta sino que se re-ejecuta.

Una ventaja de esto es que una serie de reglas puede ser utilizada en un dominio completamente distinto de en el que se genero en lugar de estar restringidas al dominio donde se desarrollaron.

3.3.4.1. Reparticularización

Esta técnica de adaptación derivacional no opera al contrario que las anteriormente descritas sobre la solución de un caso, sino sobre el método que fue usado para alcanzar dicha solución. Reparticularizar hace referencia al proceso de intercambiar un paso del proceso que llevo a la solución para utilizarlo en el contexto actual.

EJ. El programa MEDIATOR es un programa que intenta resolver por inteligencia artificial los conflictos de disputas entre dos partes. Un ejemplo simple sería que MEDIATOR intentará resolver el conflicto entre Egipto e Israel sobre el control de Sinaí.

- Ventajas:
  - Al no ejecutarse sobre la solución sino sobre el proceso que lleva hasta ella es optimo cuando el procedimiento que lleva a la solución es realmente importante para alcanzar el objetivo deseado.

- La inteligencia que genera tiene muchos mas datos para funcionar que los tipos anteriores.

- Desventajas:
  - La potencia de la reparticularización esta limitada por la potencia de planificación del sistema
  - Usa una cantidad de recursos tanto de memoria como de espacio en disco muy superior a los otros algoritmos.

- Programas que lo usan:
  - MEDIATOR
  - APU (Bhansali 1991) (4).

Capítulo 4

Aproximación a la introspección en planificación basada en casos

Aceptar nuestra vulnerabilidad en lugar de tratar de ocultarla es la mejor manera de adaptarse a la realidad.

David Viscott

4.1. Introducción

Este capítulo pretende ser una explicación de los distintos parámetros y funciones que han sido evaluadas durante la tesis, y de cómo pretenden hacer que los sistemas de razonamiento basado en casos puedan mejorar y evolucionen de forma positiva con objeto de satisfacer los cada vez más complejos requisitos que se les ponen.

Con objeto de ello se dividirá el capitulo en dos grandes partes, la primera los distintos puntos que se han encontrado de fallo y que se pretenden resolver por medio de la introspección. En este apartado se tratará tanto el modo en el que pueden fallar como el porqué de los mismos y la manera en la que nuestro sistema pretende solucionarlo. Tras esto el capitulo se centrará más en detalle en los algoritmos implementados, su funcionamiento, como sirven para evaluar los requisitos que se han planteado antes y de qué manera podrían usarse para mejorar el entorno en el que nos movemos tal y como aparece en (20) y (12).

4.2. Parámetros de error

Los parámetros de error que vamos a tratar son principalmente tres, el primero, será la falta de similitud entre los casos bases y el caso objetivo, y dentro de esto puede deberse a varios motivos, el fallo a la hora de almacenar o recuperar, la imposibilidad de tratar el caso objetivo según los parámetros necesarios o la falta de casos válidos en el sistema son algunos de los ejemplos del mismo de los cuales ya hablaremos con más detalle en los apartados que continúan. Tras esto nos iremos al caso de que la adaptación no sea correcta, esto se puede deber a que los casos de la base de datos no tienen los suficientes casos para que se pueda perfilar de forma correcta la situación del mundo, también puede haber fallado alguna precondición necesaria para la puesta en marcha del punto en concreto, y otros muchos parámetros. Por último nos queda la idea de que la similitud sea buena, la adaptación sea correcta y sin embargo el caso funcione mal, esto también es un error que puede deberse a que los casos no sean correctos, a que falten casos a introducir o haya demasiados redundantes, a que el sistema CBR tome como importante algo que no lo es y a muchas otras razones que se analizarán.

También en ambos apartados explicaremos que puede haber fallado y como comprobar si es ese el problema, así como qué preguntas se pueden plantear de cada fallo y como un sistema experto debería tratar dichas anomalías con objeto de minimizar el daño que le puedan hacer a la aplicación.

4.2.1. Fallos de similitud.

Los fallos de similitud son muy importantes a la hora de decidir si un sistema CBR es correcto o no, como ya se ha visto en apartados anteriores hay muchas formas de elegir cual de tus sistemas va a ser el utilizado y cual no, incluso puedes elegir entre varios para usarlos de forma conjunta partiendo de unos prerrequisitos para cada uno de los tipos.

Por ello es importante dar una herramienta que mida que similitud se obtiene en cada caso y como afecta la base de casos a esta similitud, para ello se puede plantear una introspección que nos intenta responder a las preguntas de ¿Hasta qué nivel se puede mejorar la similitud? ¿Qué similitud máxima se puede alcanzar de forma normal con cada algoritmo? ¿A partir de que momento la base de casos solo nos lleva a una ralentización del servicio de recuperación? ¿De qué forma se pueden mejorar estos valores? Se puede ver en este párrafo que las preguntas que surgen son muchas y variadas y que es importante intentar responder al mayor número posible con objeto de mejorar los sistemas de inteligencia artificial en el ámbito del razonamiento basado en casos.

4.2.1.1. Como aprovechar la introspección en problemas de similitud

La introspección nos permite comprobar qué resultados da un algoritmo de forma empirica basándonos en sus resultados y objetivos. En este caso nos vamos a basar en dos cosas principalmente.

- Similitud de objetivos.
- Similitud de casos.

La similitud de objetivos se centra en conseguir que un objetivo que se plantee encuentre en la base de casos del sistema un plan con el mismo enunciado de éxito, de esa forma se puede tener un plan que satisfaga el objetivo y asegurar la calidad del resultado. Esto permite eliminar los planes con fallos o que no siempre obtengan un resultado decisivo para el sistema, es decir, no es importante si tenemos tres planes A, B y C satisfacer B si lo que nos han pedido es satisfacer B, salvo claro en caso de que B sea prerrequisito para C. Nuestro planteamiento con respecto a este enunciado y tras un estudio de los algoritmos que hay en mercado nos lleva a decidir que este caso en concreto no es muy importante a la hora de analizar ya que la gran mayoría de los sistemas que hay actualmente en mercado y de la totalidad de los que hemos planteado o plantearemos a lo largo de la tesis lo satisfacen en la totalidad de los casos. Por otro lado la similitud de casos es más complicada de conseguir, y se basa en un intento de equiparar los distintos planes que tienen como objetivo el target del caso recibido para satisfacer. Para ello es importante mirar las precondiciones que tiene cada plan, ver cuáles son factibles y cuáles no, ver lo similar que es el mundo que rodea al plan que se consiguió con el mundo que se tiene actualmente y otro centenar de pequeñas aproximaciones que hacen que cambie todo el sistema.

Por ello se plantea la idea en este trabajo de monitorizar a un nivel cercano la similitud de casos exponiendo claramente que plan se consiguió recuperar, que objetivo tenía y que similitud alcanzó. De esta forma un experto podría utilizar esta información de la manera que mejor considerase, para ello se creará una consola que permitirá monitorizar estos valores y facilitará su tratamiento.

4.2.1.2. Qué utilidad tiene la introspección en lo relativo a la similitud

La introspección no solo nos es útil con objeto de validar que nuestro sistema está planteado de forma correcta, como ya se ha dicho varias veces, el uso de sistemas CBR nunca alcanzará un óptimo ya que siempre habrá algo que mejorar, ya sea el conocimiento del experto que nos puebla la base de casos, los algoritmos de almacenamiento, el calculo de similitud o cualquiera

de las variables. En el caso de la similitud se espera que el sistema marque de forma clara y concisa que resultado a obtenido, así como que planteaba conseguir con ese plan, tambien es importante tener en cuenta si lo logró o no y otros muchos parametros que hacen esta técnica dificil de implementar a nivel práctico. Empero, una aproximación a la realidad, por muy somera que sea nos puede ayudar en muchos aspectos del desarrollo de aplicaciones con inteligencia artificial que usen sistemas CBR. Algunas de las respuestas mas importantes que se pueden contestar son las siguientes:

- Esta la base de casos suficientemente bien poblada.
- Utiliza el motor CBR la formula apropiada a la hora de calcular la similitud
- Como de diferente tiene que ser el mundo objetivo comaparado con el mundo de la base de casos para que el sistema empiece a fallar.

En resumen, como se puede ver hacer una introspección del sistema CBR en cuanto a la similitud puede conllevar grandes ventajas a la hora de decidir si nuestro entorno es válido para lo que pretendemos.

4.2.2. Fallos de adaptación.

Los fallos en similitud son importantes a la hora de evaluar si nuestra base de casos esta bien poblada, pero una vez que ya tenemos la respuesta a la similitud es importante conocer cómo funciona la adaptación en nuestro sistema.

Ya se ha visto en capitulos anteriores que al igual que en la similitud existe una amplia gama de decisiones a tomar dependientes de la situacion y la aplicación exacta de nuestra inteligencia.

4.2.2.1. Como aprovechar la introspección en problemas de adaptación

La adaptación es un proceso que conceptualmente se entiende de forma bastante simple, haciendo un resumen es el proceso que desde un caso que se hizo anteriormente y que se parece más o menos al caso objetivo actual modifica el primero de los mismos de forma que se pueda aplicar a la situación en la que es necesario ahora mismo.

Sin embargo, una vez dicho esto llegamos a la idea clara de ¿Cómo sabemos que un caso ha sido bien o mal adaptado? La idea básica es que un caso bien adaptado llevará a una satisfacción del objetivo y un caso mal adaptado no, sin embargo, esto no es cierto en sistemas que no son deterministas, es decir, cuyas acciones no son siempre posibles de llevar a cabo, y por lo tanto es importante saber si el fallo se debe a la problemática de la adaptación o si bien se debe a otros motivos que no se hayan podido controlar.

Para ello se plantea conseguir sacar la información del plan que se consiguió tras la búsqueda del caso más similar tal y como se vio en el apartado anterior, y tras esto proceder a su evaluación, esta evaluación conlleva ver en qué porcentaje consiguió adaptar el plan, es decir, ¿Cuántas de sus precondiciones se cumplían? ¿Los objetos que había en el plan antiguo se pudieron emular en el nuevo? Con la respuesta a esta pregunta un experto podría conseguir algunas de las siguientes cosas:

- Modificar el algoritmo de adaptación si los casos muy similares no se pueden adaptar bien
- Descubrir quéobjetos se parecen a la hora de hacer adaptaciones y cuáles no.
- Introducir reglas y modificaciones al planteamiento del sistema de forma que la forma de almacenar la información

4.2.2.2. Problemas que tiene la introspección en lo relativo a la adaptación

Ya se ha visto en el párrafo anterior que la introspección puede ayudar en gran medida a la mejora de los sistemas CBR, pero hay un problema, y en este caso, al igual que en muchos otros, ese gran fallo es la intervención del experto, la mayoría de las veces que se plantea una introspección sobre los sistemas CBR, esta introspección asume que el experto primero, va a estar siempre disponible para preguntarle, y va a saber siempre explicar lo que el sabe de forma intuitiva a un ordenador, y segundo, el experto tiene obligatoriamente que entender la información y diagramas que se le muestran.

Estos dos aspectos son, a grandes rasgos, la mayor problemática de una introspección con idea de mejorar los sistemas CBR, en nuestro código y planteamiento, se intentará evitar en la medida de lo posible, algunos de los problemas que acabamos de mencionar, intentando hacer esta introspección lo más amigable posible al experto.

4.2.3. Fallos de conocimiento del entorno.

Los fallos de conocimiento del entorno se deben a que el sistema piensa que lo está haciendo todo bien (tiene buena similitud y adapta bien principalmente) y luego sin embargo no es capaz de alcanzar sus objetivos. Este problema es en sí mismo la mayor prueba para la introspección en este ámbito, ya que es difícil tener pruebas concluyentes salvo un simple no “alcanza el objetivo”.

Por ello, en este ámbito lo único que se puede hacer es intentar dar una idea lo más lógica posible del entorno para la rápida recepción de cualquier

información nueva que pueda ayudarnos y un proceso simple por el cual se pueda, ya sea por medio del método de prueba error, o por la prueba de varios tipos de grupos de casos con el soporte de un experto y con objeto de conseguir que el sistema tenga, gastando la mínima cantidad de recursos posibles, el máximo conocimiento del entorno del que seamos capaces de proveerle. Teniendo en cuenta siempre la relación calidad-coste.

4.2.3.1. Como aprovechar la introspección en problemas de conocimiento del entorno

La introspección como ya se ha comentado en problemas de conocimiento del entorno está más centrada en los casos y menos centrada en la modificación del código que las otras dos adaptaciones que hemos visto previamente. Sin embargo, ha dado lugar a muchas teorías y métodos para encontrar huecos en la base de casos e intentar que el coste de localizar un caso válido sea lo más barato posible.

4.2.3.2. Utilización de la introspección en los problemas de conocimiento del entorno

Ya hemos hablado mucho de los usos de la introspección y está claro que su principal labor en los ámbitos en los que nos movemos se encuentra en mejorar en la medida de lo posible todo lo que rodea la inteligencia artificial basada en sistemas CBR. Esto incluye intentar hacer lo más fácil posible al experto la interactuación con el sistema CBR.

Capítulo 5

Experimentos Realizados

Son vanas y están plagadas de errores
las ciencias que no han nacido del
experimento.

Leonardo Da Vinci

5.1. Introducción

Inicialmente este trabajo se planteó en el contexto del concurso de StarCraft organizado por Expressive Intelligence Studio en la Universidad de California, Santa Cruz, en el 2010.

El objetivo era medir la ventaja que suponía el uso de sistemas CBR y del mismo modo encontrar qué fallos tenía y qué situaciones se podían mejorar en su programación, ya fuese cambiando el código o usando una tecnología alternativa fusionada con este entorno para solventar algunos de los problemas que se descubriesen. En concreto se ha visto este planteamiento aplicado a Darmok2.

Se ha implementado parcialmente la solución propuesta debido a las limitaciones de tiempo que conlleva un trabajo de master.

5.2. Dominio Starcraft

StarCraft es un juego de estrategia en tiempo real de Blizzard Entertaiment, tiene una ambientación fantástica en la que tres razas tratan de destruirse mutuamente mientras aumentan sus tropas, sus edificaciones y van desarrollando tecnologías poco a poco. El juego, desarrollado en 1998 tuvo muchísimo éxito y se expandió por todo el mundo generando torneos a través de internet y muchísimas ventas, además ha sido actualizado durante diez años con parches y mejoras hasta llegar a la versión 1.16.1 que puede ser usada en interfaces de programación para insertar inteligencia en el juego.

Como en la mayoría de los juegos de este estilo, en StarCraft el jugador controla un ejercito de una de las tres razas (terran (humanos), protoss (una raza tecnológicamente avanzada hecha prácticamente en su totalidad por robots o armaduras cibernéticas) y Zerg, (una raza con forma de insecto y mente colmena)) , cada una de ellas con unas ventajas y unas desventajas, y su misión será destruir a uno o varios de los ejércitos enemigos que están en el mismo escenario y que son controlados por el ordenador o por otro humano, el jugador al inicio de la partida contará con un centro de mando (una edificación que crea las tropas básicas o recolectores) y cuatro de estas unidades encargadas de conseguir los recursos necesarios para poco a poco ir creando mas recolectores, edificios y unidades tanto de ataque como de defensa.

El éxito del juego se debe en parte a que rompe el concepto de piedra papel o tijera de otros juegos de estrategía en el que las unidades son o bien muy similares todas o bien fuertes unas contra otras en un bucle en el que si sabes contra quien te enfrentas puedes fácilmente luchar en una ventaja inicial muy considerable. En su lugar StarCraft consigue nivelar las razas evitando este concepto por medio de la mejora constante de las unidades para que sus fortalezas o debilidades sean cubiertas en gran parte por la estrategia utilizada y del mismo modo cada raza tiene varias formas de ganar eficientes y en cualquier momento puede cambiar de una a otra, dando tanto versatilidad a la hora de jugar como una imposibilidad para prever como serás atacado o como se defenderá el rival.

La Inteligencia de la máquina en este juego es bastante obvia y puede ser vencida una vez que se ha jugado suficiente sin mucha dificultad, ya que cada raza tiene, como ya hemos dicho, una serie de estrategias predefinidas, y a que la IA, al inicio de la partida selecciona una de ellas y la sigue durante todo el juego, por ello es bastante predecible y fácil de derrotar.

5.3. Arquitectura implementada

Para poder probar la arquitectura que se ha planteado es necesario tener una forma de enviarle ordenes a StarCraft y que él las entienda y las ejecute, para hacer esto se ha usado el ChaosLauncher, una aplicación que sirve como intermediario y que permite cargar varios pluggins al juego, así mismo, también da la posibilidad de inyectar librerías DLL al sistema tal y como se ve en (19). Ademas para poderse comunicar con el juego se hace uso de una API escrita en C++ que permite compilar librerías que, inyectadas en el juego a través del ChaosLauncher son capaces de dar órdenes y modificar el sistema.

BWApi, es una interfaz de programación completa, usable y bien documentada para el propósito de esta tesis, ya que además de poder controlar el estado del juego también se puede visualizar toda la información de la

Figura 5.1: StarCraft.

partida, tanto a nivel de unidades como de objetos en juego, permite así mismo saber que situación hay en todo momento de cada cosa en la partida y controlar a tu gusto todas las propiedades de ese objeto (incluyendo por ejemplo aumento o disminución de la cantidad de materiales que se extraen de un recurso dado o el nivel de armamento de una unidad determinada.

En la figura 5.2 , se puede ver como se conectan los modulos de Chaos-Launcher BWApi y StarCraft, el cuadro de IAUsuario.cpp es la inteligencia creada por el usuario que se encarga de todo lo que sea control del estado del juego ya sea para su lectura o para su control. Además, este código se compila en una DLL que posteriormente se le inyectará al ChaosLauncher de forma que este a su vez informe a StarCraft, con ello se consigue que cuando StarCraft comience una partida el módulo se arranque y la IA comenzará a ejecutarse.

5.3.1. Comunicación en java

El problema en este entorno se nos da cuando intentamos insertar Darmok2 (que está escrito en Java) en el entorno que hemos descrito, ya que el BWApi no nos permite trabajar con este lenguaje, además el uso de C++ conlleva varios problemas, entre los que destaca la posibilidad de que la DLL inyectada en el StarCraft interfiera en el espacio de direcciones del juego, lo que supondría una ralentización o incluso caída del mismo.

Esto se arregla haciendo que la DLL implemente un socket que se encargue de copiar el estado del mundo y enviarlo al exterior, haciendo con ello posible crear código en cualquier lenguaje y compilarlo junto con la DLL

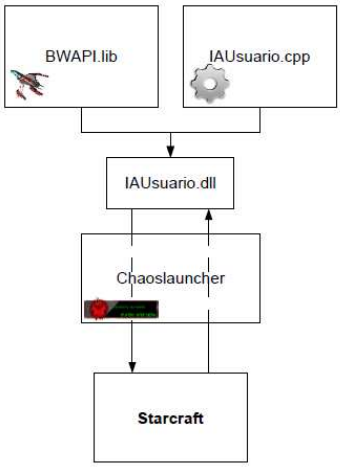

Figura 5.2: Integración de BWAPI con StarCraft

para provocar cambios en el control del juego de forma externa a la DLL.

Para esto se hizo uso de una versión mejorada de Proxy Bot (21) que implementa la arquitectura planteada en la figura 5.3 .

De forma que la DLL lee el estado del juego y envía las instrucciones pero en realidad es la IA del usuario generada en Java, con la ayuda del proxybot la que se encarga de generar la inteligencia.

5.4. Trabajo realizado

Este proyecto nació de un intento de entender cuáles eran los huecos que se podían encontrar en la inteligencia de StarCraft con el objetivo de solventar en la medida de lo posible los mismos mediante la modificación del sistema implementado, ya fuese modificando el conocimiento de Darmok2 (ampliando o reduciendo casos del CBR) o mediante la implementación de otro sistema de decisión como paso en la tesis de Ricardo (7) que intento introducir árboles JBT en el sistema de inteligencia.

Para conseguir este objetivo se han diseñado dos entornos, el primero de ellos pretende poner un escenario simple, sin ningún tipo de modificación exterior a la inteligencia y que muestre cómo se comporta esta en decisiones que se han de tomar de manera muy rápida y en entornos poco variantes, para ello se ha establecido un mapa cuadrado sin nada salvo un grupo de marines contra otro, sin ninguna mejora en los marines y sin ninguna diferencia entre los bandos, solo la maquina contra la IA que generamos.

En el segundo decidimos hacer un escenario lo más complejo posible en

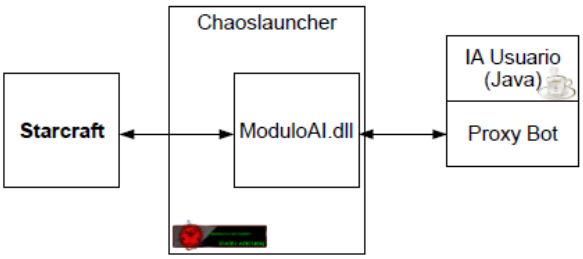

Figura 5.3: Proxy Bot para la comunicación Java con Starcraft

el que se pudiese comprobar cómo reacciona StarCraft a estrategias de más largo plazo, y que conllevan muchas más decisiones que tomar, pero para las que no hay un tiempo tan acuciante como en el sistema anteriormente visto.

Para ello planteamos una batalla en juego completo iniciando los dos bandos con terran de forma que se evitaban las posibilidades de que una raza fuese más rápida comenzando que la otra y esto introdujese ruido en nuestros experimentos, de igual forma, se habilitarón todas las tecnologías y todas las opciones del juego con objeto de hacérselo a Darmok2 lo más complicado posible a la hora de tomar decisiones.

Cabe destacar que en ambos experimentos la raza elegida ha sido Terran y que la enseñanza a Darmok ha variado como se explicará en los apartados dependiendo de lo que se quería conseguir demostrar en el experimento.

5.5. Experimento I

En este experimento vamos a comprobar cómo se comporta Darmok en un entorno pequeño y controlado con decisiones rápidas. Para ello pondremos a competir a nuestra IA contra un escenario pre-generado que potencie este tipo de decisiones. Tras esto mediremos los resultados y almacenaremos sus valores, guardando además los resultados en varias repeticiones de forma que se pueda preguntar a Darmok qué estaba haciendo en cada momento y este te responda de manera lo más clara posible. Además obligaremos a Darmok2 como ejemplo de sistema CBR en un entorno que no le es conocido. Es decir, para el que no se comporta demasiado bien (micro-management)

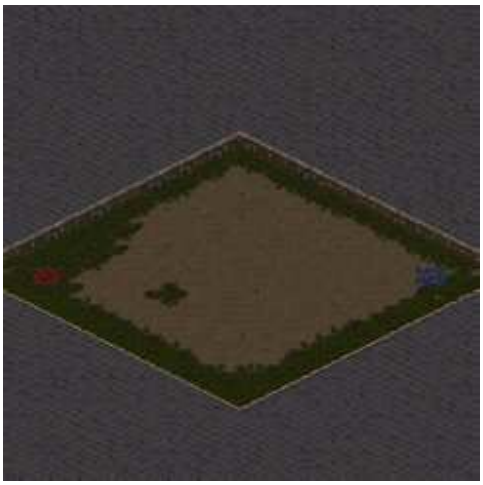

Figura 5.4: Mapa del primer experimento

5.5.1. Mapa elegido

El mapa elegido para la primera confrontación es el que se muestra en la figura 5.4, es un mapa que nunca termina y que se juega una y otra vez cada vez que un bando es aniquilado. En él, se ha intentado disminuir el tipo de objetos que pueden ayudar o dificultar la toma de decisiones de Darmok, evitando cualquier objeto que no sean los simples contendientes, eso quiere decir que no hay materiales, edificaciones animales o modificaciones del terreno de ninguna clase.

Según el patrón del primer estilo de torneo definido por la competición de StarCraft para marines, además se ha modificado este escenario de forma que la aparición de un animal 5.6 indica la desaparición de todos los contendientes de uno de los bandos y fuerza al juego a reiniciarse para una nueva batalla, cambiando únicamente un contador que se sitúa en los minerales del juego y que indica el número de rondas hechas hasta el momento por el sistema tambien cabe destacar como se ven en la figura 5.5, que además de el contador en los minerales que mostramos Darmok tambien indica (con objeto de identificar mejor los momentos erroneos de la batalla una lista de las decisiones que toma en cada momento)

Por último cabe destacar que en este mapa no se almacena quien gana o quien pierde, esto se debe a que en este documento no se trata de analizar como de buena es la inteligencia diseñada con respecto a la de StarCraft, sino que se pretende comprobar cuales son los fallos de esta inteligencia comparándola consigo misma y no con el juego para intentar mejorarlos posteriormente, su evaluación por ello se establece en función de la inteligencia

Figura 5.5: Proxy Bot para la comunicación Java con Starcraft

artificial en parametros de similitud y adaptación.

5.5.2. Inteligencia artificial

En esta sección intentaremos explicar la inteligencia desarrollada para esta prueba. Antes de nada y de igual forma que se planteaba en la sección anterior, hay que tener en cuenta que no se ha intentado maximizar la inteligencia, ya que el tamaño en memoria y la capacidad de la CPU impiden hacer pruebas correctamente si se sobrecarga la memoria de casos base.

Para empezar, debemos indicar que para hacer tus propias pruebas, tan solo tienes que dirigirte al apéndice de esta sección, en el que se indica como generar la inteligencia paso a paso.

Por nuestro lado, la IA usada, ha sido diseñada luchando 20 partidas contra el sistema (de las cuales el usuario ha ganado 14, es decir, el setenta por ciento del total) y ha perdido el resto. Esta IA tras ser cargada en el sistema genera un árbol bastante simple de decisiones que se espera sea suficiente para el entorno, también se han hecho pruebas con 50 e incluso con 100 partidas seguidas, pero los resultados variaban poco y se ha optado por la de 20 por ser la más rápida de conseguir caso de que alguien más quiera probar el sistema diseñado.

Por su parte Darmok carga la IA y genera los objetivos que ha de perseguir y como perseguirlos cuando el juego empiece, dado que no hay muchas posibilidades el motor debe conformarse con las estructuras más básicas y los objetivos más fáciles como son:

Figura 5.6: Visualización de como termina una ronda del experimento I

- Objetivos:
  - WinGoal: Este es el objetivo obvio de Darmok, y que solo se satisface cuando se gana la partida. Además es el único objetivo que se puede usar en este entorno ya que el resto (contruir unidades, contruir edificaciones, hacer mejoras) están eliminados por la concepción del entorno, recordemos que aunque no se almacene quien gana y quien pierde cada ronda dentro de una partida, Darmok2 entiende como victoria el satisfacer esta condición, lo que implica que en ausencia de otros objetivos este es el que intentará cumplir y para ello intentará asimilarse a las partidas guardadas que el entiende como victorias.
- Acciones:
  - Moverse: Esta acción es obvio que puede ser ejecutada siempre, implica la capacidad de mover una unidad por el terreno, es posible usarla.
  - Atacar Unidad: Esta acción es la más utilizada en el experimento, indica que Darmok da la orden a una unidad de atacar a otra, hay que tener en cuenta que la complejidad de esta acción es bastante importande debido a la necesidad de adaptar bien los parámetros y elegir de forma adecuada el soldado al que atacar, ya sea por distancia hasta ti, por alcance o por número de combatientes atacándole. Darmok intenta descubrir que se intentaba hacer en ese

	A	B	C
1	similaridad	adaptacion	tiempo
2	-1,30943312	100	12,432
3	-1,3046072	100	1,7600002
100	0,34924043	73,085205	6,54599
101	-1,38219837	73,28245	2,3940125
102	-1,37504476	73,47681	1,1340027
103	-1,36913869	73,66837	1,7729797
104	-1,36170216	73,857185	1,7820129
105	-1,35444195	74,04331	1,7850037
106	-1,34777365	74,2268	1,7949829
107	-1,18995973	74,38596	2,558014
108	-0,87576511	74,499565	3,3129883
109	-0,59192031	74,56847	1,8150024
110	-0,12903672	74,4239	4,097992
111	0,32147475	72,567345	6,549011
112	-1,38295595	72,74713	2,4779968
113	-1,3757537	72,924576	1,0759888
114	-1,36975501	73,09973	1,7870178
115	-1,36218189	73,27263	1,7649841
116	-1,35478509	73,44332	1,7709961
117	-1,34792689	73,61185	1,7660217
118	-1,18724191	73,758865	2,5329895
119	-0,89019144	73,86542	0,00500488
120	-1,0323633	81,40374	2,66511

Figura 5.7: Tabla que indica la similitud, la adaptación y el tiempo medio de cada plan llevado a cabo por Darmok2

ataque y lo emula (de la mejor forma que puede)

5.5.3. Resultados

En esta sección se van a explicar los resultados que se consiguieron,así como la explicación de por qué estos valores son así. Para ello mostraremos tres valores, el primero el valor de similitud que da el sistema Darmok a los planes, es decir, como de similar es el plan más parecido que encontró a la situación dada, luego el nivel de adaptación que se llegó a conseguir, lo que nos indica cómo de bien pudo convertir el mundo anterior en el que se encontraba la orden en el nuevo, y por último cuánto tiempo tardo en encontrar el plan y hacer la adaptación. Hay que tener en cuenta que Darmok, busca una serie de planes variable, pero siempre elige al final el que mayor similitud tiene y luego comprueba si puede adaptarlo, por lo que se dará el valor más alto de similitud y su adaptación tras todo el proceso y no el inicial.

Antes de empezar pedimos que se vea la figura 5.7que enseña los resultados que se obtuvieron de los tres valores, el archivo Excel al que pertenecen contiene todas las fases por las que ha pasado Darmok en el combate por lo que en la última línea se ha añadido la media para su mejor entendimiento.

El experimento ha sido hecho durante 5 minutos de forma consecutiva.

5.5.3.1. Similitud

- Resultado obtenido: El resultado obtenido fue una similitud media de -1.03 de media, en un calculo que debería resolverse en un valor entre 0 y 1 ya que es una medida en la que 0 significa nada en comun y 1 totalmente exacto. En los siguientes parrafos explicaremos que es lo que conlleva este valor y porque se da.

- Explicación de los resultados: El resultado que se ha obtenido en similitud como ya se ha visto se sale de lo esperado a primera vista en la aplicación. Esta anomalía es normal y lógica y una vez planteado el proyecto era muy probable que saliese por lo que se dejo con objeto de probar la teoría. En concreto esta similitud se debe a la forma en la que Darmok2 está programado, este entorno al ser un sistema orientado a StarCraft y en concreto a batallas completas cuando calcula la similitud tiene en cuenta las unidades que existen en el mapa (Edificios, Minerales, Equipos...), visto lo cual el algoritmo que hace para el calculo de la similitud se ve afectado por la ausencia de todos estos parametros y da un valor que se sale de rango.

  Sin embargo y puesto que Darmok2 utiliza una formula basada en la similitud por objetivos y una vez comprobados los objetivos la similitud por estado del juego, estos valores aunque equivocados, dan un plan que es válido para solventar la situación del juego. Y que de hecho es adaptado con éxito absoluto como se puede ver en la figura 5.7.

  Cabe indicar por último que esta peculiaridad ha sido remitida a Santiago Ontañon creador de Darmok2 y está estudiando su posible solución y las connotaciones que puede tener para el sistema su adaptación frente a las mejoras que pueda reportar.

- Conclusiones: Darmok cuando se trata de escenarios cerrados, se comporta de forma pobre en el encuentro de semejanzas, esto se debe a que la similitud de una estrategia a bajo nivel en la que solo se puede atacar o moverse sin edificaciones ni recursos ni gran parte de las capacidades de Darmok2 no es apropiada para el motor que se utiliza y que se encarga de las estrategias de forma más genérica, como intentaremos demostrar en el siguiente experimento realizado.

5.5.3.2. Adaptación

- Resultado obtenido: El resultado obtenido fue una adaptación del 81 por ciento, este porcenaje indica el numero de acciones y subplanes que engloban cada plan y que se pueden emular en el actual caso, lo cual era más o menos lo esperado.

- Explicación de los resultados: Como habíamos comentado en el motor de inferencia de Darmok2 se hacen dos adaptaciones estructural y paramétrica, la paramétrica que se encargaba de convertir los parámetros de las funciones para adaptarlos de forma apropiada se encarga de cambiar el ID del marine enemigo original por el nuevo, elegir al marine que vamos a usar y todo tiempo de adaptación de parámetros, cosa que se hace fácilmente y luego se hace la adaptación estructural, que se encarga de adaptar la estructura de las reglas que hay que aplicar y que a realmente no tienen demasiada complicación ya que están tremendamente restringidas (ganar partida /moverse/atacar)

- Conclusiones: Darmok es capaz de adaptar de forma adecuada las estrategias que usan pocos parámetros y que no tienen demasiadas opciones, a pesar de que tengan una similitud baja los algoritmos que se usan para adaptar dan un buen resultado como se pretende, haciendo que si bien es posible que la inteligencia no sea la mejor que pueda tener el programa, el tiempo que se le da al sistema para decidir cada plan (alrededor de 2 segundos) es suficiente para adaptar la casi totalidad de las acciones de los planes que se plantean en ese tiempo.

5.5.3.3. Resultado final

El objetivo de este experimento era tratar de ver como se encargaba Darmok2 de situaciones en las que no tiene practicamente tiempo para decidir como puede ser el combate entre dos fuerzas, intentando además evitar en la mayor medida posible las decisiones estratégicas más complejas, tras los experimentos se puede llegar a la conclusión de que el sistema se comporta relativamente mal.

Si bien es cierto que las adaptaciones son buenas el conocimiento del entorno es bastante pobre y la similitud de los planes muy negativa, por lo que en general las decisiones las toma bastante aleatoriamente. Además en otras pruebas realizadas con el mismo mapa y distintas inteligencias algunas de ellas del orden de diez veces más pesadas que la propuesta el resultado es siempre el mismo. Y la victoria queda más allá de cualquier esperanza en estos entornos.

El hecho de que el conocimiento del entorno sea bajo, nos lleva a pensar que a pesar de que la adaptación es correcta, no tiene suficientes nociones

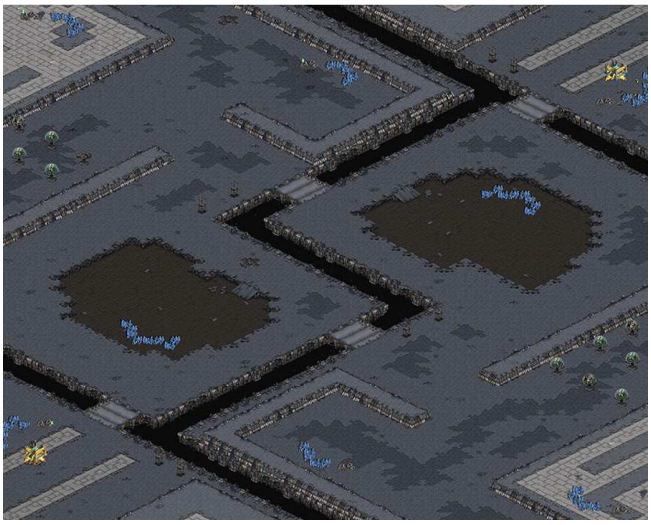

Figura 5.8: Mapa del segundo experimento

para que este dato sea demasiado relevante, ya que le falta conocimiento para hacer dichas adaptaciones.

5.6. Experimento II

En este segundo experimento se pretende comprobar como se comporta Darmok2 en un entorno de juego completo en el que tenga que usar todas sus capacidades y generar estrategias teniendo en cuenta muchos parámetros y generando con ellos un plan complejo. Hay que tener en cuenta que debido a como almacena Darmok2 los casos (ocumpan mucha memoria),es muy difícil generar planes muy largos por lo que solo pretendemos comprobar si la estrategia planteada al inicio de la partida es buena.

5.6.1. Mapa elegido

El mapa elegido es Astral Balance, un mapa relativamente simple de juego completo en el que se ha seleccionado que se juegue una partida de todos contra todos, el escenario sigue todas las reglas necesarias para un combate de estas proporciones, ya que hay suficientes materiales, animales y objetos en el terreno para que Darmok2 reconozca que esta jugando en un mapa completo y que la similitud mejore comparada con el ejemplo anterior. Además este mapa solo se juega una vez y cuando se termina la batalla no se reinicia de la forma que lo hacía en el experimento anterior.

	A	B	C
1	similaridad	adaptacion	tiempo
2	0,32576708	80	8,45
3	0,32576708	84,210526	4,0740004
4	0,32825217	69,94818	195,96199
5	0,31713914	69,55333	203,08401
6	0,32713268	69,581055	2,8919983
7	0,32713268	71,181335	0,00598145
8	0,32519847	74,07907	69,077995

Figura 5.9: Resultados de la batalla a gran escala

5.6.2. Inteligencia artificial

La inteligencia artificial generada por Darmok2 en este ejemplo usa prácticamente todas sus capacidades, tiene todos los objetivos disponibles, todas las acciones y puede generar todos los planes completos de la forma en la que se ha planteado su diseño. De esta forma se espera que Darmok use todo su potencial en cuanto a decisiones.

Por desgracia el uso de una lógica tan compleja quedaba fuera del alcance del ordenador en el que se estuvieron haciendo las practicas y para hacer esta prueba se tuvo que usar uno mas potente y además reducir la cantidad de trazas que recibía Darmok haciendo que aprendiese tan solo de dos ejemplos de partidas ganadas y solo de los 5 primeros minutos de dichas partidas.

De esta forma se esperaba que al menos Darmok2 aprendiese a iniciar una partida de forma apropiada y se viese cuales eran los mayores fallos del sistema al generar estrategias a gran escala.

Por ultimo cabe indicar que la inteligencia artificial que se entrenó fue en varios escenarios, una batalla se hizo en el que se utiliza y otra en el mapa llamado Blood Battle y que este archivo fue cortado tras su generación para poder hacer la prueba de la que estamos hablando, por lo que a pesar de ser solo los cinco primeros minutos toda la información referente a la victoria de la partida y al proceso la tiene y esto no merma en ningún caso su adaptación de planes durante la primera parte de la partida.

La elección de hacer planes en dos mapas distintos se debe a que se pretendía que Darmok2 entendiese de forma un poco más completa el sistema de funcionamiento, y no se basase tanto en las posiciones, ya que en los primeros casos de prueba, Darmok2 solo empezaba a funcionar si el lugar inicial de la partida (que se decide aleatoriamente) era el mismo donde empezaba en las pruebas.

5.6.3. Resultados

En este apartado se expondrán y analizarán los resultados obtenidos, así como una breve descripción del modo en el que funciona Darmok2 partiendo de los valores obtenidos por el experimento.

Seguimos como es lógico el mismo esquema que en el experimento anterior, pero en este momento añadimos un nuevo apartado debido a que surge al ver este experimento una nueva cosa a tener en cuenta que no se podía plantear con el anterior y que hemos llamado Conocimiento del mundo, este apartado hara referencia a lo bien que sabe interpretar el mundo Darmok2 es decir, tiene suficiente información para saber que es lo necesario para ganar una partida, el ejemplo mas claro es que si se enseña al sistema a jugar con una raza y se le pone a jugar con otra será imposible por mucho que la situacion sea igual y sea capaz de adaptarla a las nuevas tropas que sepa si lo que esta haciendo es o no lo indicado para ganar, es decir, no sabe suficiente del mundo para poder decidir por mucha información que tenga.

5.6.3.1. Similitud

- Resultado obtenido: Como se puede ver en la tabla 5.9 los resultados dan aproximadamente una similitud de 0.3 tal y como más o menos debería ser, ciertamente son un poco bajos, aunque también es cierto que esto es relativamente lógico ya que el inicio de la partida de un jugador siempre es más o menos parecido y además al haber niebla en este escenario al contrario que en el del experimento anterior, no tienes al rival para generar modificaciones ni positivas ni negativas en la estrategia inicial.

- Explicación de los resultados: Los resultados como se han comentado son los esperados, Darmok2 consigue reconocer medianamente bien las partidas similares, además, tengamos en cuenta que sólo se le ha tratado con dos mapas distintos y una partida en cada mapa, eso hace que la información que tenga no sea ni mucho menos perfecta, lo que a su vez lleva a este valor. Recordemos a su vez que esto es una probabilidad matemática, es decir, una similitud del 0,3 dice que conoce un caso que es exactamente un tercio de parecido al que se encuentra, sumando tropas, minerales, enemigos, niebla e incluso los objetivos que se querían satisfacer en dicho plan.

- Conclusiones: Darmok2 esta diseñado para jugar a un juego complejo y estratégico y no para participar en escaramuzas, por lo que al contrario que en estas los valores de similitud indican que este sistema es muy bueno estratégicamente hablando y que es capaz de convertir la información obtenida tanto del juego como de sus bases de casos en acciones que son similares a las que se dieron en situaciones anteriores

(al menos antes de la adaptación).

5.6.3.2. Adaptación

- : Resultado obtenido: El resultado obtenido es si cabe mejor que el esperado, además de adaptar de media el 74 por ciento de los casos, el sistema coge tanto casos de una como de otra inteligencia de forma adecuada, y nada mas empezar envía a sus recolectores a investigar la zona y a recoger materiales mientras crea nuevas unidades de forma apropiada.

- Explicación de los resultados: Los resultados obtenidos son mas o menos los esperados por las razones antes señaladas, Darmok, al ser un sistema más basado en la estrategia a gran escala que en la estrategia a pequeña escala, tiene más puntos de apoyo para decidir, además tiene todas sus capacidades y puede generar los planes de forma adecuada y usando los subplanes apropiados, esto lleva a que si bien la generación de un plan se alarga hasta valores excesivos (más de un minuto para cada adaptación de media) estos planes son complejos y completos y se forman de forma apropiada a la situación que tiene delante y al entorno concreto en el que la puede aplicar.

- Conclusiones: Darmok2 adapta de forma correcta los planes tanto a nivel paramétrito (como se demostraba con mayor facilidad en el experimento anterior), como a nivel funcional, tanto en sistemas en los que el mundo es muy parecido como en otros en los que es mas diverso. Por último cabe destacar que el hecho de que la adaptación sea correcta debería indica, sumado al hecho de que la similitud es aceptable que Darmok2 si que es un sistema útil para la manipulación de juegos de estrategia en tiempo real.

5.6.3.3. Conocimiento del mundo

Se puede ver en este experimento que Darmok2 hace una similitud aceptable del problema y adapta los sistemas de forma adecuada, sin embargo, no le es posible vencer una partida y esto crea las dudas obvias de esta situación. ¿Qué esta pasando en Darmok2? Las opciones estudiadas son las siguientes:

- Funcionamiento incorrecto de Darmok2: Esta opción es lógico plantearla, a fin de cuentas como explicábamos en el estado del arte Darmok2 y cualquier sistema CBR se ve obligado a tomar varias decisiones a lo largo de su creación como pueden ser:

    - Qué forma de calcular la similitud se va a usar

- Qué forma de adaptación se va a usar
- Sobre qué parte del entorno tendremos control y que información obtendremos

Además de esto hay que tener en cuenta que el tiempo que tarda en adaptar un plan depende de la cantidad de información que este contenga y además de la cantidad de planes contenidos en la base de casos, y como se puede ver en la figura 5.9el tiempo que tarda en responder Darmok2 al sistema es demasiado largo y sus planes se terminan de adaptar mucho después de que hayan sido planteados.

Por otro lado, la cantidad de pruebas hechas con darmok2 tanto por esta tesis como por Ricardo, o por el propio Santiago Ontañón nos llevan a decir que el sistema es suficientemente bueno para su tarea y por lo tanto el propio planteamiento de que el sistema este fallando por inconsistencia del mismo es imposible. Aunque si que hay que tener en cuenta que el tiempo de respuesta es muy bajo para lo que se necesita. Lo que de nuevo nos lleva a que la idea probablemente es añadir otra tecnología que ayude a Darmok2 a hacer algo, aunque este algo sea mucho menos complejo, mientras piensa y una vez que Darmok2 tenga el plan este le tome el relevo para terminar de ejecutar las acciones pertinentes.

- Conocimiento muy vago del entorno: También otra posibilidad es que Darmok2 no conozca el entorno, aunque sinceramente y con lo que sabemos del sistema y de la estructura que se ha montado precisamente para conocer el entorno podríamos decir sin miedo a equivocarnos que el sistema tiene todo el conocimiento sobre el entorno que necesita para tomar decisiones, en este caso, cuando nos referimos a entorno nos estamos refiriendo solamente a su capacidad para saber lo que hay en el juego no en la base de casos.
- Poco conocimiento de sus opciones: Este caso indica la posibilidad de que la cantidad de trazas generadas para entrenar a Darmok2 sea insuficiente, esta opción es con mucho la causa más probable para el fallo de Darmok2, la potencia del ordenador en este experimento se queda corta a la hora de analizar las trazas que emite el sistema, pero aun así, tarda demasiado tiempo en decidir sobre sus opciones, por lo que podemos concluir que con una máquina más potente y un programa auxiliar que le ayude a decidir de forma rápida mientras Darmok genera la estrategia se podría llegar a un resultado competitivo en este campo.

5.6.3.4. Resultado Final

Este experimento tenía principalmente dos objetivos, el primer era comprobar la validez de un sistema CBR y en concreto de Darmok2 como sistema

de planificación para juegos de estrategia en tiempo real, con todo lo que esto implicaba, generación de estrategias, adaptación de partidas anteriores basadas en esquemas no deterministas, calculo de similitudes por medio de algoritmos generados a través de información sacada de StarCraft por medio de una aplicación auxiliar, y todo lo demás que es necesario para manipular un juego de estas características, en eso claramente Darmok2 es totalmente suficiente por sí mismo, tiene un algoritmo robusto para calculo de similitudes, siempre y cuando estemos hablando de estrategias a gran escala y un sistema de adaptación más que válido para el cometido que se le pide.

Además también se quería comprobar si el sistema tenía algún tipo de fallo que pudiese ser solventado ya fuese por un humano o por un programa auxiliar, y al final en este experimento hemos llegado a la conclusión de que darmok2 a estos aspectos tiene dos grandes fallos:

- Lo primero de todo es la mejora del algoritmo de similitud y adaptación, hay que ver si es posible mejorar su versatilidad para conseguir que funcione en entornos como el del primer experimento, asegurando de esa forma que puede responder a todas las situaciones que se le planteen, también hay que ver si se puede mejorar el conocimiento general del mundo, ya sea por medio de la inyección de casos o por medio de la eliminación de aquellos trozos de la memoria que no aportan información útil.

- Lo segundo y probablemente más importante es conseguir que Darmok2 tenga un conocimiento apropiado del entorno, lo que le permitiría entender mejor los casos con los que se maneja, adaptar de forma más conveniente los sistemas y tardar menos en todo el proceso de generar las acciones que enviar al juego

  En el apartado de conclusiones explicaremos que nos aportan estos resultados y por donde pensamos seguir el estudio y modificación de este sistema.

5.7. Funcionalidad de repetición

Para que el sistema se pueda replicar y ver los resultados propuestos, se ha diseñado una clase Java llamada ConsolaEstadisticas, en esta clase se implementa un sistema que permite, juntando una repetición de StarCraft y uno de los archivos de salida xml que genera nuestro sistema visualizar las decisión que tomó Darmok2 en cada momento, es decir, con objeto de ver cuando Darmok2 tardó demasiado en pensar algo o bien cuando llego a una conclusión equivocada se ha diseñado un sistema que permita regenerar partidas después de haberlas jugado.

Con esto en mente se puede ayudar mucho a solventar los problemas

Figura 5.10: Ejemplo de repetición de partida

que vimos en Darmok2, es decir, si bien recordamos que los problemas eran principalmente tres:

- Darmok2 no es capaz de encontrar partidas que sean muy similares en escenarios cerrados y falla a la hora de tomar decisiones rápidas en estos entornos, es decir, es imposible para Darmok2 funcionar con toda su potencia cuando no tiene objetivos o acciones validas para el caso objetivo.

- Daromk2 gasta demasiada memoria a la hora de generar un plan completo y es necesario ponerle poca información pero de mucha calidad si se desea trabajar de forma correcta con él.

- Darmok2 tarda de media demasiado en generar los planes y debido a ello el juego le encuentra demasiado poco desarrollado para que encuentre en él un oponente digno como pretendemos que sea. Además, debido a esto mismo el sistema no tiene mucha versatilidad a la hora de planificar.

En nuestro proyecto se pretende solventar en la medida de lo posible este problema sin usar ninguna herramienta más aparte de Darmok y para ello nuestro sistema genera esta clase que como hemos dicho repite las partidas de forma sistemática y mostrando las decisiones que se tomarón en Darmok, la similitud que dio cada plan, como fue adaptado y que objetivo pretendía, de la forma en la que se ve en la figura 5.10que representa una partida llevada a cabo con este sistema.

5.7.0.5. Código

Como se ha explicado el código de esta Consola nos ayuda a ver lo que ha pasado en Darmok, para ello nos hacen falta algunas cosas:

- Repetición de StarCraft: Lo primero que necesitamos es una repetición de StarCraft en la que hayamos jugado.
- el archivo xml generado a través de la partida que se jugó en esa repetición.

Una vez que tenemos esos dos objetos ponemos el archivo xml en la carpeta de resources del proyecto ConsolaEstadistica y lanzamos la repetición. En este momento se nos lanzará la ConsolaEstadistica y creará una ventana en la que se podrá preguntar en cualquier momento como va l situación en StarCraft y que hacia Darmok2 en ese momento, como se puede ver en la figura 5.10

5.7.0.6. Objetivo

El objetivo de este código se basa en que un jugador experto pueda ver lo que hizo Darmok2 de forma simple en cada partida y pueda poblar a Darmok con la información que necesita para mejorar, de forma que se especialice más en la toma de decisiones y no se cargue especialmente la memoria, insertando solo aquella información que necesita el sistema para resolver sus fallos de forma apuntada, y no metiendo nada que ya este tratado y arreglado por la información previamente obtenida, así mismo se puede sustraer de Darmok2 toda información que le engaña y le hace pensar que una decisión es correcta cuando en realidad está equivocada.

Como podemos ver, esta funcionalidad es muy útil ya que puede ahorrar muchos problemas y está planteada para ser un método de ayuda a un jugador humano experto para que interactúe con el sistema de forma lo mas simple posible a la hora de decidir que información quiere insertar en la base de casos del sistema CBR

Capítulo 6

Conclusiones y trabajo futuro

La recompensa del trabajo bien hecho es la oportunidad de hacer más trabajo bien hecho

Jonas Edward Salk

6.1. Conclusiones

Nuestro trabajo trataba de hacer una introspección a los sistemas CBR, aplicándolos en concreto a los juegos de estrategia en tiempo real, un ámbito en el que su uso está siendo cada vez más complejo con objeto de alcanzar, durante la realización de la tesis además, se ha generado un código en java que facilita en la medida de lo posible la intervención de un experto en la mejora y desarrollo de las inteligencias artificiales, en el proyecto, enlazado con Darmok2 se ha generado un proyecto java portable a otras tecnologías de similar configuración que muestrea los sistemas de IA basados en CBR.

El trabajo, además cuenta con una aproximación cercana al estado del arte actual y a las tecnologías que ahora mismo se están planteando para la resolución del tipo de problemas que se han tratado a lo largo de la tesis, sin dejar de lado su ámbito práctico. En la tesis se pretendía generar un código que ayudase en la medida de lo posible a la mejora de los sistemas de inteligencia artificial. Y en este aspecto ha sido todo un éxito y se ha probado por medio de la eliminación de casos redundantes que se puede mejorar un sistema CBR, queda en el tintero hacer la prueba al revés, generando una inteligencia mejor o que dure más tiempo en un caso práctico. Sin embargo, el descubrimiento de algunas vulnerabilidades del sistema que se ha usado deja prueba suficiente para concluir que el trabajo es útil y puede ayudar a los sistemas en los que se aplique.

Por otro lado el estudio introspectivo es un área que puede ser mucho más explotada de lo que lo es actualmente y con esta tesis también se pretende dejar un precedente para todas aquellas que se hagan posteriormente, de

cómo este tema puede tener suficiente relevancia y dar un apoyo real a los sistemas que se generen tanto dentro de los sistemas CBR como fuera de los mismos.

Por último también era objetivo de la tesis la comprobación del sistema Darmok2 y atestiguar su validez como un sistema CBR competitivo en desarrollo de inteligencia para juegos de estrategia en tiempo real (en este caso StarCraft) y se cree que este entorno es perfectamente capaz de conseguir, a la larga, todas las metas que se le han propuesto.

6.2. Trabajo Futuro

Como hemos visto en esta tesis han quedado varios puntos abiertos algunos de los cuales espero poder retomar en un futuro. De ellos los más importantes son los siguientes:

- Desarrollar con la ayuda de un experto un ejemplo que demuestre la utilidad del código desarrollado para mejorar StarCraft.
- Continuar con la implementación de árboles JBT para que junto con él punto anterior se cree una inteligencia competitiva.

Apéndice A

Empezando con Darmok2

A.1. Introducción

En este capítulo se detalla la manera de montar toda la estructura que ha sido necesario plantear para poder hacer este proyecto, su objetivo es hacer más fácil a cualquiera que venga detrás y que necesite plantearse como empezar con este sistema los pasos a seguir para conseguir montarlo sin tener que pasarse semanas buscando tutoriales y leyéndose documentación al respecto.

Para empezar debemos tener en cuenta que hay que tener varias cosas:

- El código fuente de este proyecto
- StarCraft + BroodWar + el parche 1.1.16 de broodwar
- ChaosLauncher
- BWApi 2.6.1 (versión exacta)
- Eclipse, Netbeans o cualquier otro compilador de java

Antes de empezar quiero indicar que las imágenes que se muestran en este manual están hechas con Eclipse 3.4 y que pueden no coincidir con la versión que utilice el usuario.

A.2. Carga de proyectos

La Carga de los proyectos debe ser hecha en el entorno Eclipse o similar de forma individual, para ello nos vamos a importar y seleccionamos la carpeta o carpetas donde tenemos los fuentes de la forma en la que se ve en la imagen A.1.Si hay cualquier error debemos asegurarnos de que los Path están bien puestos, para ello daremos en el proyecto que tenga una X roja con el botón

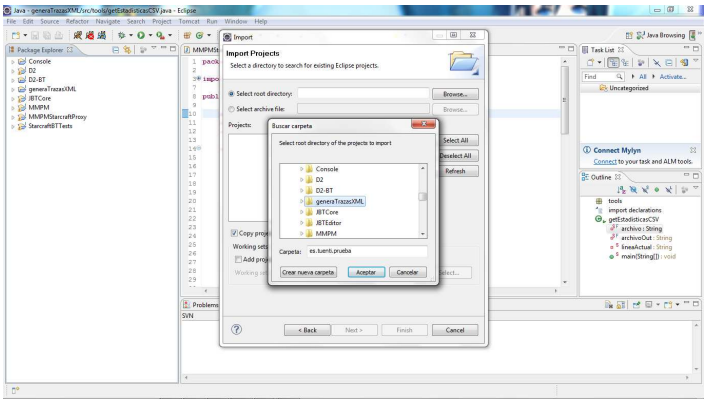

Figura A.1: Importar proyectos

derecho y luego nos pondremos en la pestaña Build Path y añadiremos los proyectos necesarios como se ve en esta lista:

- Console : ninguno
- D2: generaTrazasXML + MMPM
- D2-BT: Console + D2 + JBTCore + MMPM
- generaTrazasXML: ninguno
- JBTCore: ninguno
- MMPM: ninguno
- MMPMStarCraftProxy: MMPM + D2
- StarCraftBTTests: todos

Con esto debería funcionar la aplicación, para comprobarlo nos vamos a StarCraftBTTester y lanzamos la clase StarCraft/MMPMStarCraftProxy, vemos que funciona bien.

A.3. Instalación StarCraft, BroodWar y parche

Para instalar el StarCraft hay que insertar el disco de StarCraft y luego seguir las instrucciones, posteriormente instalar el disco de BroodWar y de

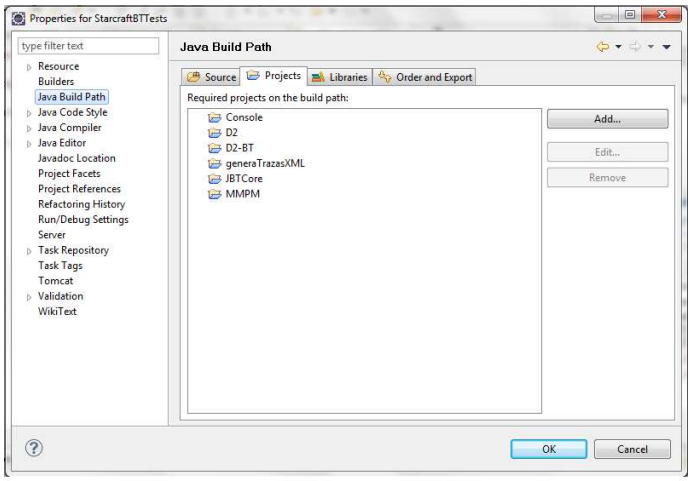

Figura A.2: Crear Path de compilación

nuevo continuar con las instrucciones para acabar instalando el parche 1.1.16 para BroodWar (es necesario tener este parche porque ChaosLauncher a la hora de inyectarse al sistema requiere de librerías que se le proporcionan por medio del mismo)

A.4. Instalar ChaosLauncher y BWApi

EL ChaosLauncer una vez instalado tan solo necesitará ser ejecutado lanzando el exe que le acompaña, si todo esta correcto y el BroodWar ha sido instalado bien debería tener el aspecto de la imagen A.3, en concreto deberían estar seleccionables las mismas casillas que se ven en la imagen.

Por su parte el BWApi tiene varias carpetas que hay que recolocar, siguiendo el nombre que tengan hay que ponerlas de la siguiente manera:

- ChaosLauncher: se debe pegar en la carpeta que contenga el programa del mismo nombre.

- StarCraft: se debe pegar en la carpeta de StarCraft, en esta carpeta se insertarán las DLL que contengan la clase C++ con el proxy para que Darmok2 pueda funcionar.

- Windows: La carpeta de Windows debe ser puesta en la carpeta del sistema operativo del mismo nombre.

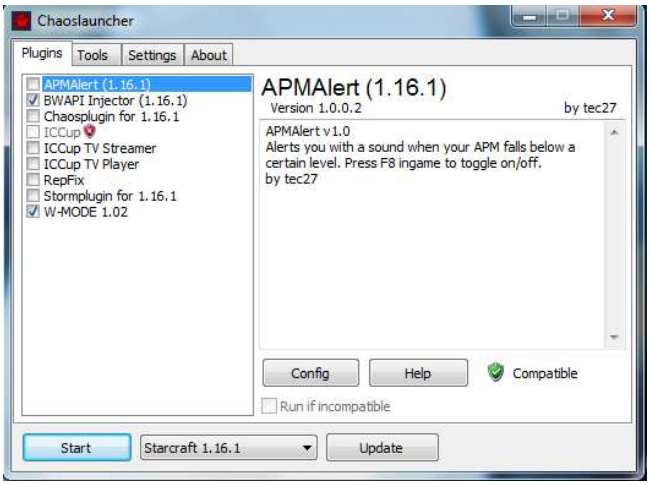

Figura A.3: Crear Path de compilación

A.5. Primera prueba

Una vez montado el entorno hay que añadir la DLL al sistema para que Darmok pueda interactuar con el juego, para ello cogemos la dll que se encuentra en el proyecto en la sección StarCraftBTTest/C++/proxy y lo metemos en la carpeta StarCraft/bwapi/AI, luego nos vamos al archivo bwapi.ini que se encuentra en StarCraft/bwapi y comprobamos que esta la siguiente línea:

- $ai_dll = bwapi - data/AI/ExampleAIModule.dll$

Una vez hecho eso podemos iniciar el StarCraft, iniciar el eclipse y lanzar el StarCraftMMPMProxy, cargar una partida y ver como juega nuestro sistema, aunque por el momento depende del escenario y las capacidades del ordenador solo seremos capaces de ver que StarCraft ha conectado con el proxy y si tenemos suerte como algunos recolectores se mueven, para seguir deberemos crear nuestra inteligencia y cargarla.

A.6. Inteligencias ya cargadas

En la situación actual hay unas cuantas inteligencias que pueden ser cargadas sin esfuerzo y que pueden servir para probar el sistema, estas están cargadas en variables de entorno en la clase StarCraftMMPMProxy, y lo único que hay que hacer es iniciarlas, poner el Proxy Asociado correcto e iniciar el escenario. Para hacerlo basta con poner uno de los siguientes packs

en los espacios comentados del principio del sistema. (Recordemos que ya hay seguro un pack puesto y habrá que comentar el que este:

- pack Marine Battle
  - $privatestaticfinalConfigurationconfiguration = Configuration.MARINESBATTLE$;
  - $privatestaticfinalStatsTypestatsType = StatsType.NORMAL$;
  - mapa: Master/MarineBattle
- Batalla con Bunkers
  - $privatestaticfinalConfigurationconfiguration = Configuration.BUNKERS$;
  - $privatestaticfinalStatsTypestatsType = StatsType.BUNKERS$;
  - mapa; Master/Bunker
- Batalla entre vultures
  - $privatestaticfinalConfigurationconfiguration = Configuration.VULTURES$;
  - $privatestaticfinalStatsTypestatsType = StatsType.VULTURES$;
  - mapa Master/Vultures
- Batalla a gran escala
  - $privatestaticfinalConfigurationconfiguration = Configuration.TERRAN_BATTLE$;
  - $privatestaticfinalStatsTypestatsType = StatsType.BASIC$;
  - mapa: Astral Balance

Recordemos que los mapas que están en el master vienen en la carpeta del proyecto testMaps en StarCraftBTTests.

Bibliografía

[1] A. AAMODT and E.PLAZA. Case-based reasoning: Foundational issues, methodological variations, and system approaches. arti?cial intelligence communications. In *Case-based reasoning: Foundational issues, methodological variations, and system approaches. Arti?cial Intelligence Communications*, 1994.

[2] Richard Alterman. An adaptive planner. In *AAAI'86*, pages 65–69, 1986.

[3] Kevin D. Ashley. Reasoning with cases and hypotheticals in hypo. *International Journal of Man-Machine Studies*, 34(6):753 – 796, 1991. Al and Legal Reasoning. Part 1.

[4] Sanjay Bhansali. Domain-based program synthesis using planning and derivational analogy. *AI Magazine*, pages 31–33, 1991.

[5] Blizzard. http://us.blizzard.com/blizzcon|blizzcon.

[6] María Guadalupe Olivares Conde. Razonamiento basado en casos, reutilizar. revista de informática fedora 2014, 2011.

[7] Ricardo Juan Palma Duran. Extendiendo darmok, un sistema de planificacion basada en casos, mediante arboles de comportamiento. In *Extendiendo Darmok, un Sistema de Planificacion Basada en Casos, mediante arboles de Comportamiento*, 2010.

[8] Gonzalo Flórez-Puga, Marco A. Gómez-Martín, Pedro P. Gómez-Martín, Belén Díaz-Agudo, and Pedro A. González-Calero. Query enabled behaviour trees. *IEEE Transactions On Computational Intelligence And AI In Games*, 1(4):298–308, November 2009.

[9] RL Simpson Jr. A computer model of case-based reasoning in problem solving: An investigation in the domain of dispute mediation, 1985.

[10] Igor Jurisica. Representation and management issues for case-based reasoning systems, 1993.

[11] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7:358–386, March 2005.

[12] David B. Leake. Experience, introspection and expertise: Learning to refine the case-based reasoning process. *J. Exp. Theor. Artif. Intell.*, 8(3-4):319–339, 1996.

[13] Team Liquid. http://wiki.teamliquid.net/starcraft/chaoslauncher.

[14] Dana S. Nau, Yue Cao, Amnon Lotem, and Hector Mu noz-Avila. Shop: Simple hierarchical ordered planner. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 968–975, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[15] Miguel Ángel Casado Hernández. Integración de conocimiento web en sistemas de razonamiento basado en casos. In *Integración de Conocimiento Web en Sistemas de Razonamiento Basado en Casos*, 2010.

[16] Inés Mendiz Noguero. Un estudio sobre la aplicacion del razonamiento basado en casos a la construccion de programas. In *Un estudio sobre la aplicacion del razonamiento basado en casos a la construccion de programas*, 1992.

[17] Oscar J. Prieto, Juan J. Rodríguez, Carlos J. Alonso, and Aníbal Bregón. Identificación de fallos en sistemas dinámicos mediante stacking y alineamiento dinámico temporal. In *XI Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA 2005*, pages 103–111, 2005.

[18] BWAPI Team. http://code.google.com/p/bwapi/.

[19] D2 Team. Darmok2: http://sourceforge.net/projects/darmok2/.

[20] Patrick Ulam, Ashok Goel, and Joshua Jones. Reflection in action: Model-based self-adaptation in game playing agents. In *In D. Fu J. Orkin (Eds.) Challenges in Game Artificial Intelligence: Papers from the AAAI Workshop (Technical Report WS-04-04*. AAAI Press, 2004.

[21] Ben Weber. http://eis.ucsc.edu/starproxybot.

[22] Ben G. Weber and Michael Mateas. Case-based reasoning for build order in real-time strategy games. In *Case-Based Reasoning for Build Order in Real-Time Strategy Games*, 2009.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

Autorización

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Introspección en sistemas de planificación basada en casos aplicados a juegos de estrategia", realizado durante el curso académico 2010-2011 bajo la dirección de Pedro Antonio González Calero y con la colaboración externa de dirección de Antonio A. Sanchez Ruiz-Granados en el Departamento de Ingeniería del Software e Inteligencia Articial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Fdo: Fernando García González

En Madrid a 09 de setiembre de 2011