UNIVERSIDAD DE ALCALÁ

Escuela Técnica Superior de Ingeniería Informática

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE GRADO

Estudio y desarrollo de IA para juegos estratégicos

GUILLERMO REMÍREZ GONZÁLEZ

2015

UNIVERSIDAD DE ALCALÁ

Escuela Técnica Superior de Ingeniería Informática

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado
Estudio y desarrollo de IA para juegos estratégicos

Autor: Guillermo Remírez González
Director: Antonio Moratilla Ocaña

TRIBUNAL

Presidente: .....

Vocal 1º: .....

Vocal 2º: .....

CALIFICACIÓN: .....

FECHA: .....

Índice general

Capítulo 1

Resumen

En este TFG se plantea la posibilidad de diseñar una inteligencia artificial capaz de efectuar un razonamiento táctico eficiente que desafíe al usuario en el entorno estratégico de un videojuego 2D. Dada una serie de entes, normas y suposiciones, se estudiarán distintas combinaciones de algoritmos de tal manera que un hipotético jugador en el mencionado videojuego encontrara cierta dificultad al enfrentarse a dicha IA en igualdad de condiciones. Para ello se analizará la utilidad de distintos algoritmos de teoría de juegos y programación eficiente en dicho entorno, todo ello sin olvidar las limitaciones de recursos existentes en el entorno dado.

Capítulo 2

Summary

In this TFG we lay out the possibility of designing an AI capable of making an efficient tactical reasoning in a way that challenges the user on a strategic 2D videogame environment. Given a series of entities, rules and assumptions, we will consider different combinations of algorithms in such a way that an hypothetical player would find some level of difficulty when facing said AI on a level playing field. For that purpose, we will analyze the usefulness of different algorithms from game theory and algorithmic efficiency in such environment, always keeping in mind the resource limitations imposed by it.

Capítulo 3

Palabras Clave

Inteligencia Artificial, videojuego, Game Maker, estrategia

Capítulo 4

Introducción

La inteligencia artificial se perfila como una de las disciplinas más complejas y abstractas del siglo XXI. Su aproximación a la mayoría de las definiciones que, habitualmente, se utilizan para detallar la inteligencia, son aún hoy en día pobres y lejanas [2]. Es por ello que, al enfrentarse a distintos problemas en los cuales el uso de un razonamiento metódico y un conocimiento previo son indispensables, es inevitable la realización de un diseño previo y concienzudo de un sistema que pueda simular dicho razonamiento. No existe, en el presente, una inteligencia artificial general capaz de enfrentarse a una variedad de problemas similar a la que la mente humana se enfrenta día a día. Teniendo esto en mente, el diseño de dicho sistema será, en la mayor parte de las ocasiones, específico y centrado únicamente en el entorno de ese problema en concreto.

El objetivo de este TFG no es otro que el de, aceptando esta realidad, diseñar un sistema de inteligencia artificial capaz de tomar decisiones razonadas y estratégicamente válidas en un entorno previamente definido de forma arbitraria. Esta declaración de intenciones, algo vaga y genérica, queda detallada al mencionar el entorno en el que se trabajará: Un videojuego de estrategia militar en dos dimensiones.

Para cualquier versado en el tema, la idea de un producto así no es nada nuevo. Hay muchos títulos en la industria del videojuego que han trabajado dicha idea con anterioridad, enfocando el atractivo del juego en la dificultad que entrañan para el usuario al ponerle en situaciones de clara desventaja frente a un enemigo que, por otro lado, hasta la fecha no ha podido alcanzar el nivel de la inteligencia humana (por otro lado, es importante señalar que, a pesar de las perspectivas de futuro, hoy en día la IA orientada a los videojuegos no comparte los mismos objetivos que su contraparte más tradicional [11]).

Dichos títulos han intentado distintos acercamientos al diseño de un contrincante que suponga un reto para el jugador experimentado, valiéndose del uso de distintos scripts y algoritmos complejos para analizar el entorno y evaluar las distintas opciones con las que poner en "jaque.ªl jugador humano. Es difícil negar que este estudio se encuentra sino inspirado, al menos sí influenciado, por esa clase de juegos.

Se podría afirmar que el objetivo principal de este estudio es crear un sistema completo que combine una serie de algoritmos y módulos de forma efectiva, implementándolos en el producto final (el videojuego) de tal manera que se satisfagan ciertos requisitos básicos, como una ejecución dinámica y rápida, un uso de memoria eficiente y, por encima de todo, un comportamiento de las unidades controladas por la IA (CPU player) que simule de la manera más efectiva posible un razonamiento táctico por parte de un hipotético jugador humano que compitiera con el usuario.

Esta definición algo difusa sugiere una serie de disciplinas, relacionadas con la inteligencia artifi-

cial, que indudablemente influirán en menor o mayor medida en el progreso del estudio mencionado. También resulta sencillo suponer que, siendo un objetivo tan ambicioso, la duración y recursos propuestos no desembocarán en una solución óptima.

El ámbito en el que se ubica este estudio, pues, no es otro que el de la toma de decisiones en base a ciertos factores. Algunos de estos factores, según el diseño del entorno, podrán ser evaluados mediante heurísticas ponderables y, por tanto, comparados de forma directa y fiable. Otros de ellos, sin embargo y como se puede comprobar en el desarrollo del estudio, dependen de varias evaluaciones complejas difíciles de cuantificar de forma sencilla. Debido a esto, es necesario un estudio a fondo de las posibles soluciones y algoritmos propuestos con anterioridad para la resolución de casos similares.

Un buen ejemplo de cuando dichos factores entran en juego sería al evaluar la deseabilidad táctica de un estado dado en el juego. Una decisión que pudiera, por ejemplo, eliminar una parte significativa de los recursos del contrincante, sería altamente deseable sin tener en cuenta otros factores. Sin embargo, si al hacerlo se pusiera en peligro varios de los propios recursos que más tarde pudieran ser eliminados por el contrincante, es posible que el estado final de tomar esa decisión pudiera ser evaluado como menos deseable que el actual. La cosa se vuelve más compleja cuando se contempla la posibilidad de coordinar varias acciones aparentemente no deseables para crear un estado final que, al sumar dichas decisiones, otorgue una ventaja significativa frente al contrincante.

El videojuego sobre el que se desarrollará este estudio será diseñado acorde a una serie de reglas absolutas e inamovibles, a las que el propio usuario y la IA deberán ceñirse a la hora de ejecutar sus acciones. Con el objetivo de simplificar las pruebas que se llevarán a cabo para evaluar esta última, el videojuego ofrecerá una serie de niveles que el usuario jugador tendrá que superar. En el juego, solo se contemplará el caso de que un jugador humano compita contra el sistema. Alternativas en las que un jugador humano compite contra otro, o una IA contra otra, no amplían significativamente el campo de estudio, por lo que serán dejadas aparte en la versión final del programa.

Estos puntos de partida sugieren que, al igual que en el ajedrez y otros planteamientos tácticos de similar naturaleza, la cantidad de posibilidades que se pueden dar a raíz de las acciones que se escojan crece de forma exponencial. Analizar todas esas posibilidades para medirlas usando los factores antes mencionados tendría una complejidad exponencial que apoya la suposición previa de que la solución óptima es inalcanzable, al no poder explorar todos los nodos que aparezcan en el árbol de decisión. Lo cual señala directamente al siguiente punto a tener en cuenta: Las barreras que se presentan en el desarrollo de la aplicación.

Las limitaciones del sistema serán en todo momento teóricas, si bien se realizarán una serie de pruebas empíricas en un simulador para comprobar la efectividad de los algoritmos propuestos. Sin embargo, estas limitaciones resultan cruciales a la hora de plantearse el diseño, pues, como se esclarece en el párrafo anterior, trazan la línea en la cual tendremos que situarnos, entre una búsqueda completa de las distintas alternativas (con un coste exponencial e imposible de implementar con los medios actuales) y un juego caótico y sin planificación alguna (con un coste constante).

Además, se busca huir del comúnmente llamado "scripting": Esto es, aplicar una serie de comportamientos preestablecidos si se dan ciertas condiciones para emular un pensamiento táctico. Aunque a la larga es inevitable recurrir a ciertas evaluaciones arbitrarias, la intención de este estudio es emular lo más fiablemente posible un razonamiento lógico centrado en la evaluación de distintas acciones, no una sucesión de procesos configurados que emulen un aparente pensamiento táctico.

Con esto en mente, decir que el resultado final posiblemente será menos eficiente que algunos de los sistemas similares del mercado no es una locura: La filosofía subyacente es más cercana a un intento por llevar a la práctica ciertas ideas antes expresadas que a crear una IA capaz de emular un verdadero desafío mental.

Este segundo caso, como es lógico, no descartaría el uso de aquellos scripts para asegurarse de que ciertas estrategias habitualmente utilizadas que no estén contempladas en el sistema de razonamiento

propuesto sí pudieran ser utilizadas en contra del usuario. Jugadores expertos probablemente sabrían reconocerlas y contrarrestarla, pero el usuario medio no advertiría siquiera la existencia de dichas técnicas. Nuestro caso, por el contrario, está dispuesto a sacrificar esa efectividad a cambio de renunciar al uso de estas en lo posible.

En parte por esto, la utilización de grafos, tablas y diagramas simplificados que permitan la ejecución de dicho razonamiento es imperativo. Aunque un soporte visual resulta útil (sino necesario) para que el usuario pueda interactuar con el sistema, la IA tendrá que "diseccionar"las raíces de dicho soporte para aplicar sus propios algoritmos y, de esa manera, poder alcanzar cierta capacidad deductiva que intente asemejarse a la de su contrincante.

En lo que respecta al uso de algoritmos, la complejidad prevista del entorno hace que propuestas habituales de la inteligencia artificial deban de ser apiladas para poder encontrar un compuesto mixto que, usando dichas propuestas de forma coordinada y precisa, haga emerger una IA competitiva.

Se pueden resumir, en conclusión, los objetivos del TFG en las siguientes líneas:

1. Investigación sobre distintos algoritmos y propuestas teóricas útiles para la toma de decisiones en base a un conjunto de estados conocido de forma completa o parcial, con la presencia de incertidumbre proporcionada por un agente externo (el jugador humano).

2. Estudio, en ese mismo sistema, de posibles soluciones con las que proporcionar una capacidad de anticipación a las decisiones del usuario, razonamiento lógico y manejo efectivo de los propios recursos.

3. Conclusión final sobre el resultado del estudio, así como posibles comentarios y conjeturas a raíz de lo aprendido.

Para completar estos objetivos se utilizarán distintos recursos, como un software de creación de videojuegos y distintos artículos y libros de bibliografía usados para la documentación. Se intentará ofrecer un entorno visual agradable y entretenido para el usuario, con la finalidad de realizar varias pruebas en las que participen distintos voluntarios distintos del diseñador de la IA.

La aplicación, dotada de cierta ambientación para, de nuevo, sumergir al usuario, será creada desde cero, enfatizando sin embargo la parte técnica frente a los simples efectos visuales. Todo el trabajo artístico, más adecuado para un proyecto de diseño gráfico que para este, será tomado de otras aplicaciones y páginas de descarga en vez de ser realizado desde cero.

Inicialmente, la idea de la ambientación gira en torno a una serie de batallas militares de carácter contemporáneo, con ciertos elementos ficticios. Esta ambientación ofrece muchas posibilidades con las que retar a nuestro sistema de IA, y ofrece una serie de características que adecúan la dificultad de su diseño al nivel deseado.

Asimismo, la decisión de establecer un sistema de turnos simplifica la tarea, permitiendo un análisis más exhaustivo del entorno que en el caso de un videojuego en tiempo real (RTS). También resulta consistente con los paralelismos que se trazaron previamente con el ajedrez, aunque, a diferencia de este, cada jugador podrá usar todos sus recursos en su turno en vez de solo uno de ellos.

Una vez la aplicación esté terminada, se revisarán los resultados y el proceso que llevó a estos, extrayendo varias conclusiones y propuestas de mejora. Se hará especial hincapié en los aciertos y fallos de elegir la aproximación utilizada para la IA, que se comparará con otras aproximaciones diferentes.

Capítulo 5

Sobre los videojuegos de estrategia

5.1. Introducción

5.1.1. Qué es un videojuego de estrategia

¿Qué es un videojuego de estrategia? Una definición ampliamente utilizada hoy en día en el mercado del entretenimiento digital. No es raro ver cómo se etiquetan distintos títulos bajo esta definición, muchos de los cuales ni siquiera parecen tener un componente táctico o de planificación estratégica. En un mundo globalizado en el que cualquier persona puede realizar una crítica de un videojuego y çolgarla.[en] internet, no es raro encontrar gran cantidad de etiquetas y definiciones que a veces pueden generar debate.

Al pensar en videojuegos de estrategia, cualquier entendido pensará inmediatamente en varios títulos del género, ya sean recientes o lleven circulando por el mercado varias décadas. La posibilidad de manejar distintas unidades en una batalla mental contra otro jugador o la máquina es una idea que atrae a personas de todas las edades, y ha sido uno de los estilos más explotados por la industria del videojuego, que hoy en día ya ha investigado distintas variantes.

Muchas personas no familiarizadas con los juegos de estrategia, sin embargo, podrían encontrar ciertas dificultades a la hora de hablar de este amplio género. Resulta, por tanto, natural empezar por definirlo para establecer una serie de referencias a las que el videojuego que se va a implementar deberá ajustarse. Al intentar desglosar esta definición, sin embargo, encontramos ciertas "zonas grises"que dejan lugar a cierta interpretación. Es cierto que hay videojuegos que la mayoría identificaría con esta etiqueta sin lugar a dudas, pero muchos otros, que acogen algunas peculiaridades del género sin llegar a centrarse en él, pueden generar cierta controversia.

5.1.2. Definición

Podríamos definir un videojuego como estratégico si este pone especial énfasis en la toma deliberada de decisiones que, requiriendo cierta reflexión por parte del jugador, ejercen un impacto considerable en el estado del juego [7]. Esta definición ofrece un contraste claro frente a juegos en los que los reflejos o la habilidad son el foco principal, como podrían ser algunos *fps* o juegos de tipo arcade.

Sin embargo, también es cierto que algunos de los juegos de dichos géneros presentan una serie de características que, si bien no convierten el pensamiento estratégico en la parte central del juego, sí que le reservan cierta importancia que algunos argumentarían podría introducirles dentro de esta

definición. En un *fps*, decisiones como quedarse tras una cobertura o exponerse a ser alcanzado por los enemigos a cambio de alcanzar una posición desde la cual obtener ciertas ventajas tácticas pueden cambiar radicalmente el desarrollo del juego.

En algunos juegos, la posibilidad de dar órdenes a unidades aliadas o incluso de elaborar complicadas estrategias usando diversos recursos al alcance del jugador, manteniendo al mismo tiempo las principales características de los videojuegos *fps*, han dado lugar a que muchos los identifiquen dentro de un género híbrido comúnmente llamado *first person shooter strategy games*.

Por tanto, la definición de un juego de estrategia es algo amplio, que puede abarcar varios sectores y, por tanto, también da lugar a varios géneros híbridos.

5.1.3. Términos habituales

Cuando se habla de un videojuego de estrategia, muchas veces aparecen términos que merece la pena explicar:

1. **Estrategia**: Se refiere a la toma de decisiones, a gran escala, para alcanzar un objetivo. El estratega es aquel que plantea los hitos que se deben alcanzar para poder cumplir dicho objetivo general. En el ambiente de los juegos de estrategia, el objetivo general suele ser vencer al enemigo. La estrategia serían el conjunto de objetivos menores que se deben cumplir para vencerle, como controlar una zona, recuperar ciertos objetos o invadir una ciudad.

2. **Táctica**: Si bien es la estrategia se encarga de establecer los objetivos a alcanzar, es la táctica la que se encarga de decidir cómo conseguirlos. En el ejemplo anterior, la estrategia sería la que se encargaría de establecer que se debe dominar una zona concreta, pero es la táctica la que decide el curso de acción a la hora de completar dicho objetivo: Dónde va a colocarse cada unidad, que patrón de defensa se va a usar en caso de ataque, etcétera.

3. **Logística**: La logística se relaciona con el control de los recursos y la organización de estos para poder llevar a cabo las distintas tácticas establecidas. En el ejemplo anterior, sería la encargada de asegurarse de que las unidades estén bien abastecidas, que los beneficios que pueda ofrecer dicha zona sean almacenados de forma correcta, etcétera.

4. **Niebla de guerra**: La niebla de guerra, en un videojuego de estrategia, se refiere normalmente a la incapacidad de conocer la totalidad de los detalles del terreno de juego. Se suele representar con una zona oscura en su manifestación más habitual, que es la de impedir al jugador ver más allá del terreno que sus unidades pueden divisar con su capacidad de visión.

5.2. Algo de historia

5.2.1. Los primeros juegos de estrategia

Aunque el concepto de la IA aplicada a los videojuegos es algo tan antiguo como los propios videojuegos, es cierto que los primeros juegos de estrategia y táctica que aparecieron demandaban requisitos que hasta la fecha otros géneros no se habían planteado. Ya no bastaba con hacer que un enemigo siguiera una ruta predefinida, o atacara al jugador nada más verle, o siguiera ciertos patrones ajustados para el nivel del juego en el que se encontraba. Los arquitectos que estaban detrás del personaje no jugador (PNJ) se encontraban con una serie de dificultades que las propuestas previas no cubrían.

Algunos consideran el programa de ajedrez de Turing como la primera inteligencia artificial orientada a un videojuego. Sin embargo, obviando los primeros videojuegos relacionados con juegos de mesa como el "tres en raya"(tic-tac-toe), el ajedrez o similares, no es descabellado afirmar que el primer juego de táctica fue el Çomputer Bismarck"(1980). Juegos previos como Ïnvasion"(1979) y sus predecesores habían intentado simular la ambientación, pero no fue hasta este último que se integró una inteligencia artificial capaz de ofrecer un reto a un solo jugador en lugar de obligar a este a competir con otro jugador humano.

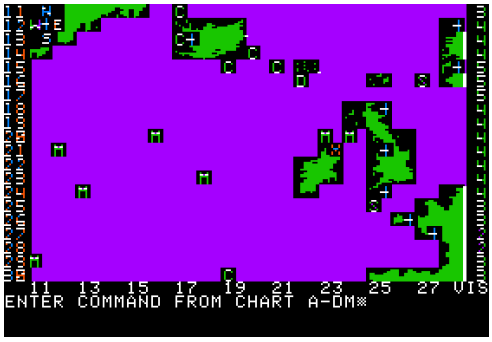


Figura 5.1: Muestra del aspecto de una partida en el Çomputer Bismarck"

Este concepto, como todo, no era nuevo. Y más tarde, como era obvio, la necesidad de una IA avanzada para juegos que poco o nada tenían que ver con la estrategia ("shooters", RPGs", juegos de plataformas, etc...) se haría obvia. El desarrollo de los PNJ seguiría varios caminos, dependiendo del estilo. En muchos casos, incluso se limitaba voluntariamente las capacidades de esta para evitar que sobrepasara al jugador. Enfrentarse a enemigos que nunca fallaban y tomaban decisiones óptimas en cuestión de milisegundos transformaba a muchos videojuegos en simplemente imposibles de completar.

Sin embargo, en el género de táctica y estrategia esas limitaciones no existían en la mayoría de los casos. El pensamiento táctico requería una serie de aptitudes que un ordenador no podía simular con facilidad: Pensamiento lateral, visión de profundidad, evaluación rápida de alternativas deseables, planificación de acciones compuestas, lógica difusa, objetivos a largo plazo... Esta diferencia esencial hacía (y hace) que existiera una gran diferencia entre el jugador y la IA.

Esto podía solucionarse, en muchos casos, incrementando los recursos de esta última para hacer los niveles más difíciles de completar. Dar al PNJ más unidades iniciales, potenciar sus valores de ataque o defensa, darle ventajas como el conocimiento completo del estado de juego frente a un conocimiento limitado por parte del jugador humano... Sin embargo, muchas veces estas soluciones eran obvias, y algunos diseñadores invertían esfuerzos adicionales en crear un sistema complejo (o simularlo) de tal manera que esta diferencia de capacidad entre la mente humana y la IA se hiciera más corta.

Si se analiza Çomputer Bismarck", se encuentra la primera aproximación a esta idea. Basándose en el algoritmo minimax, el PNJ podía examinar una serie de vías de acción y elegir aquella que le ofreciera mayor ventaja frente a su contrincante. El juego, basado en una batalla naval de la segunda guerra mundial entre las fuerzas británicas y alemanas, no presentaba una cantidad demasiado grande de opciones y, por tanto, respondía de forma natural al algoritmo.

Sin embargo, al empezar distintas compañías a producir nuevos juegos de estrategia cada vez más complejos y sofisticados, era cuestión de tiempo que la inteligencia artificial se encontrara cada vez con situaciones más complicadas de resolver. Los primeros juegos "4X"(llamados así por

los cuatro estrategias que permiten: explorar, expandir, explotar y exterminar), como el Reach for the Stars"(1983) introducían un nuevo campo de decisiones para la IA, que ya no solo tenía que decidir cuándo, dónde y cómo usar sus unidades, sino también qué unidades y mejoras adquirir. Sorprendentemente, realizaban su trabajo de forma más que eficiente, en parte gracias al intensivo proceso de desarrollo que se había dado durante la creación del juego.

Figura 5.2: Reach for the stars"fue uno de primeros juegos en añadir nuevas dimensiones a la IA

5.2.2. La evolución del género

Al finalizar la década de los 80, videojuegos como "Sim City"(1989) o Civilization"(1991) elevaban de nuevo la dificultad de la IA a nuevos niveles. El estado del juego dependía de muchas variables, y el rango de acciones posibles era tan grande que ya no bastaba con aplicar los algoritmos ya conocidos en el género: Había que adaptarlos cuidadosamente para descartar la cantidad exponencial de decisiones que presentaban según avanzaba el juego y poder decidir entre una cantidad manejable de opciones sin ralentizar el juego por ello.

Fue en esta época, también, cuando los primeros RTS (Real Time Strategy"), como Hergzog Zwei (1989), considerado el primer RTS por muchos, o el más existoso Dune II (1992), aparecieron, presentando una nueva dimensión que hasta ahora la IA no había tenido en cuenta: "Pensar.[en] tiempo real. Ya no solo era necesario tomar las decisiones en un corto periodo de tiempo para evitar que el jugador esperara demasiado: Ahora había que tomar dichas decisiones al mismo tiempo que se realizaban otras y se examinaba el campo de batalla.

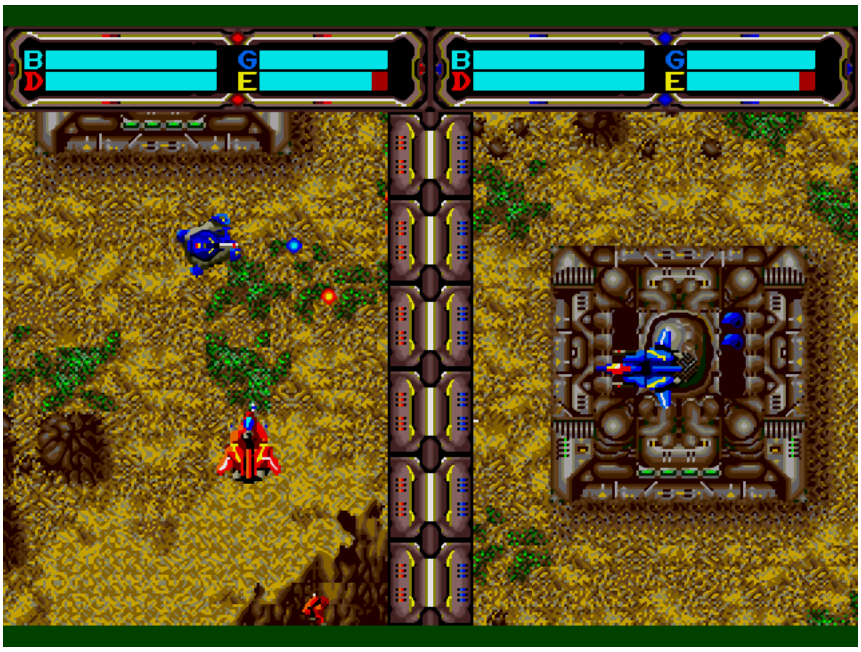


Figura 5.3: Hergzog Zwei, uno de los primeros juegos que introdujo el concepto de .estrategia en tiempo real"

Es irónico que, pese a la gran diferencia que en apariencia guardan los dos estilos, la IA conservaba la mayor parte de sus características al pasar de uno a otro. Había que adaptarla, como era de esperar, pero los algoritmos de toma de decisiones eran sorprendentemente similares, con la diferencia de que, en los TBS ("Turn Based Strategy") la toma de decisiones se realizaba una sola vez por turno, y permitía una mayor inversión de tiempo y recursos en optimizarla, mientras que en los RTS dicha toma de decisiones se realizaba de forma periódica (sino constantemente) y con mayores limitaciones.

Pronto, los RTS empezaron a hacerse más fuertes y a desplazar progresivamente a los BTS que, si bien no han desaparecido del mercado, sí que han cedido su antigua supremacía a sus habitualmente más emocionantes contrapartidas. Mientras que los segundos se centraban habitualmente en dar prioridad a los elementos tácticos y a la reflexión, los primeros añadían un nuevo elemento: La rapidez en la toma de decisiones y la habilidad para controlar el entorno con fluidez y firmeza.

Es imposible hablar de los videojuegos de estrategia sin mencionar sagas como "Warcraft", .Age of Empires", "Total War." "Starcraft". Perfectos ejemplos de cómo el género RTS se ha ido abriendo paso en la cultura general de los videojuegos hasta convertirse en uno de los principales géneros de la industria. Desde la llamada "Gran Estrategia"("Total War: Shogun 2", "Distant Worlds: Universe"...) hasta el "micromanaging"de unas cuantas decenas de unidades ("Starcraft II", .Age of Empires II"), la gran cantidad de variantes y nuevas ideas que siguen apareciendo entre los nuevos lanzamientos hace pensar que el género aún tiene una larga vida por delante.

Figura 5.4: Age of Empires, ambientado en distintos periodos históricos, se perfila como una de las principales sagas de RTS de las últimas décadas

5.2.3. IAs avanzadas

En la gran mayoría de estos juegos se puede observar una IA avanzada, capaz de hacer frente al jugador a varios niveles. Aunque una de las principales ventajas frente a su contrincante es la rapidez (mientras que el jugador debe gastar tiempo en usar la interfaz para impartir una orden a sus unidades, el PNJ solo necesita ejecutar unos comandos de forma casi instantánea), el inteligente uso de scripts y algoritmos avanzados de toma de decisiones hacen que, cada vez más, la IA sea capaz de tomar decisiones complejas para sorprender al usuario.

Los recursos que se deben manejar tampoco son escasos. Empezando con las primeras entregas de la saga "Total War", la IA pasaba de tener que controlar decenas de unidades a tener que manera cientos, sino miles, de ellas. A pesar de las curiosas y astutas soluciones que propusieron (algoritmos genéticos, conexiones ponderadas entre las unidades, contraste entre las sugestiones estratégicas y los deseos de la unidad...), todavía se podían comprobar cómo esta gran cantidad de carga de trabajo daba lugar a algoritmos que desembocaban en extraños patrones: Unidades que quedaban atrapadas en bucles, arqueros que disparaban a sus propios aliados, y otra serie de errores que, en el producto final, no afectaban a la jugabilidad pero quedaban como una pequeña broma entre los jugadores.

Figura 5.5: En Medieval: Total War, la necesidad de controlar una gran cantidad de unidades implicaba que fuera necesaria la utilización de diversas técnicas avanzadas

En "Supreme Commander"se permite y aconseja el uso de dos monitores para llevar un control del mapa global y poder controlar los cientos de recursos que el jugador tiene a su disposición. Es, por tanto, inevitable pensar que la IA tendrá que manejar, como mínimo, esa misma cantidad, probablemente más, de diferentes recursos con la dificultad añadida de no poder evaluar el estado del juego çon un simple vistazo". Son solo dos ejemplos de cómo los obstáculos que se presentaban frente a los programadores no dejaban de crecer cada vez más.

El campo de la IA en los RTS ofrece, por tanto, gran cantidad de desafíos e interesantes alternativas, atrayendo a una gran cantidad de programadores que, combinando el uso de las herramientas ya establecidas (o, en algunas ocasiones, inventando nuevas), buscan la combinación perfecta que permita a su creación soportar la inmensa carga impuesta por la creciente demanda de los usuarios del mercado, adaptarse a una gran variedad de posibilidades y combinaciones mutables y ofrecer un comportamiento adecuado al nivel de juego para dar la sensación de estar jugando contra otro jugador, y no contra la máquina. Un trabajo difícil, pero satisfactorio.

Figura 5.6: Civilization, uno de los principales referentes del género TBS

Pero es el TBS el género que más relación tiene con este TFG, y las perspectivas no son tan prometedoras. Aunque grandes títulos como los de la saga Çivilization.º "Heroes: Might and Magic"han dejado una fuerte huella y no muestran indicios de estar cerca de su final, la proporción de este tipo de juegos en comparación a otros de estrategia es claramente menor.

Capítulo 6

Requisitos del proyecto

6.1. Metodología

Antes de empezar con el diseño del sistema, es necesario plantearse qué es lo que se busca. ¿Qué objetivos debería poder cumplir el producto acabado para considerarlo adecuado a lo que se buscaba en un principio? Esta pregunta puede dar lugar a valoraciones muy subjetivas, que dependan de la persona que está respondiéndola, y por tanto no sean útiles para otro hipotético sujeto con una opinión diferente.

Es, por tanto, imprescindible buscar una serie de parámetros y directrices mediante las cuales juzgar de forma más o menos objetiva (dentro de la imposibilidad de alcanzar una objetividad completa) el resultado final, y valorar si la Inteligencia Artificial diseñada es capaz de abarcar las distintas marcas que se le habían establecido. Estas directrices deberán ser cuantificables, si es posible, o lo suficientemente neutrales como para que la opinión personal del hipotético juez que valora dicho resultado no se vea altamente condicionada por sus opiniones personales.

Sin embargo es obvio que, llegado el momento, habrá algunas características que se quieran lograr que sean esencialmente subjetivas, y que por tanto no entren dentro de esta categoría que se está definiendo. Aspectos como la jugabilidad, el aspecto o la valoración del conjunto están intrínsecamente relacionados con las preferencias de cada sujeto, por lo que necesariamente variarán en cada valoración. Estos casos se agruparán por separado para resaltar su naturaleza. Se ha decidido dejar esta serie de requisitos como relevantes, ya que aportan un tipo de valoración al juego que no se puede obtener centrándose únicamente en los otros requisitos.

Señalar también que estos requisitos pueden no llegar a ser cumplidos. Aunque esto se valorará más adelante, si llegado el producto final hay uno más requisitos que no se han cumplido de forma efectiva, no significa necesariamente un fracaso a la hora de diseñar el programa. Puede que el planteamiento no fuera el correcto, o que el requisito fuera inalcanzable con los recursos de los que se disponía al establecerlo.

Se pueden considerar estos requisitos, por tanto, como una meta a alcanzar, y no una línea de acero necesaria para completar el producto. Su principal función es guiar el diseño y la programación del videojuego, de tal manera que siempre estén en mente mientras se van integrando cada una de las partes que lo componen. Esta "brújula"servirá como referente para dar prioridad a ciertos campos frente a otros.

A la hora de establecer requisitos, es importante empezar con una idea algo más general, que luego aplicar a las ideas concretas del juego. Es por eso que se ha dividido la lista de requisitos en dos partes: Generales y específicos:

1. Generales: Representan las ideas generales del proyecto, y lo que se busca de un modo más global, sin entrar en detalles.

2. Específicos: Son una conversión de los requisitos generales para aplicar en el juego.

De esta manera, se presenta la lista de forma coherente con la evolución natural de estos requisitos desde su concepción hasta su evaluación.

6.2. Requisitos generales

6.2.1. Gráficos

Variedad

Los gráficos deben de ser variados y distintos. Se busca realizar un juego con múltiples opciones y unidades, por lo cual se deberá invertir tiempo y recursos en buscar un conjunto variado de gráficos que ofrezca diversidad al usuario. La mayoría de juegos en 2D en la actualidad cuentan con un gran repertorio de "sprites", o imágenes renderizadas, para sus distintos niveles y objetos. Aunque la biblioteca de gráficos no tiene necesariamente que ocupar una gran cantidad de espacio, se busca que cada elemento del juego esté diferenciado por una representación gráfica adecuada.

Complejidad

Los gráficos deben de ser complejos, al menos en lo que se refiere a las representaciones que ofrecen de ideas y conceptos. Este objetivo podría resultar bastante subjetivo si no se especifica correctamente, pero definiendo una línea concreta para establecer el nivel de complejidad deseado, puede concretarse como algo medible y, por tanto, objetivo.

Obviamente, no se busca un conjunto de gráficos avanzados y artísticamente notables: Este no es un proyecto centrado en la calidad artística, sino en el estudio de la Inteligencia Artificial. Por otro lado, es cierto que un juego basado en figuras poligonales o representaciones muy abstractas y vagas de lo que se está mostrando puede llegar a ser confuso y entorpecer la experiencia de juego. Aunque no hay duda de que hay juegos que con gráficos minimalistas y un poco de creatividad pueden alcanzar un alto valor de jugabilidad y una alta calidad visual, en este caso se ha preferido optar por la opción más segura, ya que los gráficos no son, como se ha explicado previamente, el foco del proyecto.

6.2.2. Jugabilidad

Variedad

El juego debe de ser variado en cuanto a sus componentes. Un juego estratégico con un tipo de unidad y un puñado de reglas sencillas puede ser profundamente complejo y ofrecer una serie de posibilidades interesantes, pero en este caso se ha decidido optar por una mayor cantidad de opciones. El usuario podrá elegir entre varios tipos de alternativas a la hora de interactuar con el juego, y contará con una serie de elementos diferenciados para poder llevar a cabo sus objetivos.

Aunque este objetivo se centra en gran parte en ofrecer una experiencia entretenida al jugador, ofreciéndole suficientes alternativas como para enriquecer el juego con más tipos de opciones, también plantea una dificultad añadida para la IA, que deberá enfrentarse a distintos tipos de situaciones en las cuales dispondrá de variadas unidades, cada cual con sus características especiales.

Duración

La duración del juego debe de ser suficientemente larga como para poder evaluar a la Inteligencia Artificial en diferentes situaciones. Un juego corto y breve podría resultar entretenido, pero no dará muchas opciones para que la Inteligencia Artificial pueda demostrar su potencial. Además, se busca que el usuario pueda alargar su experiencia de juego lo máximo posible, por lo que una buena cantidad de niveles, o incluso un editor personalizado para diseñar los propios, podría ser bastante útil.

Reglas

Las reglas del juego deberán estar definidas y serán inmutables, excepto cuando se especifique lo contrario. Además, tanto el usuario como su rival, la IA, seguirán esas mismas reglas, encontrándose en una igualdad de condiciones en lo que respecta a las alternativas de control que tienen sobre los distintos elementos del juego. Se ofrecerá al jugador un breve resumen de las reglas, que le permita conocerlas desde el primer momento. Asimismo, se integrarán dichas reglas en la IA para que esta las tenga en cuenta y se adapte a ellas de la mejor forma posible.

6.2.3. Plataforma

Instalación

El juego deberá poder instalarse sin problemas en cualquier ordenador actual, y ser accesible para toda clase de usuarios sin necesidad de que estos tengan nociones avanzadas de informática. Este requisito está directamente orientado a facilitar la experiencia de juego para toda clase de usuarios.

Requisitos técnicos

El juego deberá de contar con unos requisitos técnicos adecuados, sin que estos resulten excesivamente altos. Como medida, el juego debería poder funcionar en cualquier ordenador en buen estado que tenga unas prestaciones concretas, o superiores. Dichas prestaciones deberán ser definidas posteriormente.

Este requisito no solo tiene que ver con la accesibilidad del juego para un mayor número de usuarios, sino con las limitaciones de la IA. Se deben poder adaptar los procesos de esta a una utilización de recursos adecuada, no muy alta, pero que igualmente le permita ser eficiente en su cometido.

6.2.4. Inteligencia artificial

Herramientas teóricas

La inteligencia artificial debe apoyarse en herramientas teóricas adecuadas, y basarse en ciertos fundamentos ya establecidos del campo, como algoritmos estándar o teorías relacionadas. No es un trabajo de cero, sino un estudio de cómo aplicar lo que ya existe sobre un entorno específico y definido. Esto implica una tarea de estudio y planificación, que acompaña a la fase de análisis y diseño del juego, mediante la cual se podrá fundamentar las soluciones aplicadas sobre una base estable y reconocida.

Por supuesto, habrá que adaptar dichas herramientas de forma adecuada y extraer sus puntos fuertes para hacer de ellas algo viable y útil en el entorno propuesto. Esta parte de adaptación es

el principal atractivo del proyecto, pues es la base sobre la que se construirá la IA definitiva que interactuará con el usuario final.

Entorno desconocido

La inteligencia artificial debe de ser capaz de interactuar con un entorno desconocido y variable, que muta y es modificado por el usuario y por la propia IA a lo largo de la partida, y que guarda una serie de variables y factores aleatorios a los que la IA no tiene acceso. Esto hace que no se pueda diseñar un sistema único enfocado a un entorno cerrado, como el de una IA que operase solo en un mismo nivel en el que las condiciones iniciales siempre sean similares. El número de elementos, la disposición de estos o incluso los detalles de los objetivos a alcanzar podrían variar.

Esto, como es lógico, obliga a diseñar la IA de tal manera que esta se base en predicciones más o menos aproximadas, según los datos que tiene a mano. Estas predicciones deberán permitirle alcanzar un resultado deseado independientemente del estado inicial del juego, por lo que la capacidad de análisis y la toma dinámica de decisiones son aspectos clave que deben buscarse.

Tiempo de espera

Al ser el principal producto que se ofrece como resultado del proyecto un videojuego, este debe de reunir ciertas características lúdicas que en otros productos de software no serían necesarias. El tiempo de espera es una de las más importantes, puesto que el jugador normalmente preferirá ejercer un control activo y constante sobre el entorno en vez de esperar durante largos periodos de tiempo a que los procesos subyacentes se lo permitan.

Esto también impone la que probablemente sea la mayor tara a la que se va a enfrentar la IA: La limitación de tiempo. Cuando le toca el turno de actuar a la máquina, esta debe tener en cuenta que el tiempo de análisis y toma de decisiones con el que cuenta es un recurso limitado, pues el usuario está esperando. Por tanto, debe de ser lo más eficiente posible, aunque ello implique que no encuentre la alternativa óptima, de modo que pueda realizar sus movimientos en un periodo de tiempo razonable y devolver el control al usuario cuanto antes.

Tipos de IA

Se dividirá la IA, si es posible, en distintos tipos, de tal manera que se pueda elegir entre varios comportamientos diferenciados, cada uno orientado a un tipo de estrategia. Las diferencias entre este tipo de metodologías será observable y dará lugar a distintos estilos de juego. Estos estilos pueden ser más o menos efectivos, o adaptarse mejor a distintas situaciones.

6.2.5. Otros requisitos

Facilidad de análisis

El resultado final, es decir, el juego, deberá ser fácilmente evaluable simplemente jugándolo. La inteligencia artificial deberá demostrar sus capacidades en la práctica, sin necesidad de que el jugador tenga nociones sobre los métodos que utiliza. Además, el tiempo requerido para formar una opinión al respecto no deberá ser muy grande.

6.2.6. Requisitos subjetivos

Gráficos atractivos

Los gráficos deben de resultar adecuados y agradables para el usuario. Aunque no se busca presentar una experiencia visual única, el hecho de hacer más agradable la interfaz del juego hace que el usuario se encuentre más predispuesto a sumergirse en su dinámica, y por tanto el componente gráfico de un videojuego siempre es algo a tener en cuenta.

En este caso, dicho componente gráfico no es un fin, sino un medio para establecer el sistema de juego y el medio por el cual se interactuará con la IA, que es el verdadero protagonista de este proyecto. Esto no quiere decir que no sea importante, pues como medio es el principal conector entre las ideas abstractas aquí presentadas y el sujeto que las evalúa.

Se busca que el colorido y variedad de los gráficos hagan de la experiencia de jugar al juego algo entretenido y deseable, de tal forma que el usuario en cuestión se sienta cómodo al jugarlo y sea receptivo a repetir la experiencia. Esto se puede lograr de varias maneras: Usando un tamaño de pantalla adecuado, estableciendo imágenes coloridas y detalladas para cada objeto, cuidando el aspecto de los menús y los efectos, etcétera.

Este requisito es esencialmente subjetivo, ya que se está evaluando de forma directa la calidad artística del proyecto, y dicha calidad es por definición algo no cuantitativo, al menos de forma universal. Cada sujeto podrá valorar las opciones gráficas que tiene frente a él de una manera más positiva o negativa dependiendo de sus gustos.

Gráficos identificables

Los gráficos del juego deben de ser fácilmente diferenciables e identificables, de modo que la experiencia de juego sea sencilla e intuitiva. Esto podría parecer algo superficial en un primer vistazo, pero es en realidad una de las necesidades más importantes de todo videojuego, pues la jugabilidad depende esencialmente de ello. La experiencia de probar un videojuego en el que las representaciones de los elementos son confusas o poco claras es normalmente negativa, y por tanto se debe realizar un esfuerzo adicional para asegurar que la interfaz sea lo más clara posible.

Esto se puede lograr de varias maneras. Para empezar, los elementos han de ser fácilmente diferenciables entre sí. Dos objetos distintos no deberían dar lugar a confusión, y con un rápido vistazo deberían de ser fácilmente reconocibles. Los menús deben de estar bien señalados para que no se mezclen con el entorno, y los distintos eventos que se sucedan deben de aparecer de forma natural y progresiva, siendo fácilmente identificables.

Este requisito es, de nuevo, subjetivo: Dos personas distintas podrían estar en desacuerdo al juzgarlo, ya que lo que para uno podría resultar un entorno claro y fácil de entender al otro le podría resultar confuso, o incluso molesto.

Entorno sencillo

El entorno de juego, es decir, la interfaz mediante la que el usuario realiza sus distintas opciones, ha de ser intuitivo y sencillo, de modo que el juego se desarrolle sin necesidad de un esfuerzo adicional innecesario por parte de dicho usuario. Es fácil caer en tecnicismos o métodos complejos que dificultan la experiencia de juego y requieren una curva de aprendizaje elevada para todo aquel usuario que, sin conocer el juego, se embarque en una partida sin preparación previa. Al fin y al cabo, es fácil que el diseñador o diseñadores, que han sido los responsables de crear el juego, comprendan procesos y sistemas complejos que para ellos resultan obvios (ya que han sido los responsables de integrarlo), pero que para otros resultarían confusos o complejos.

Este punto es crítico para que la experiencia de juego sea satisfactoria. Una interfaz compleja siempre dará lugar a error, confusión y, en última instancia, frustración si el jugador no puede comprenderla. Para que el juego resulte competente y se pueda evaluar la IA sin escollos, se debe procurar que aspectos esenciales como la interacción con el entorno sean intuitivos y no den lugar a errores o malentendidos.

Este requisito es esencialmente subjetivo, debido principalmente a la variedad de jugadores que pueden presentarse: Lo que para unos sería sencillo y fácilmente entendible, para otros sería excesivamente complejo.

Desafío

El juego debe ofrecer cierto grado de desafío. Esta necesidad radica principalmente en la búsqueda de una experiencia entretenida para el jugador. Es cierto que hay juegos con un grado bajo de dificultad, pero dichos juegos suelen experimentar una vida útil algo más corta que la de otros homólogos que ofrezcan un reto mayor. Es habitual que el jugador busque un reto al enfrentarse a otro, o en este caso, a la inteligencia artificial. La confrontación con una serie de desafíos mentales y físicos, y la necesidad de esforzarse para resolverlos, habitualmente convierten la experiencia en algo entretenido.

Sin embargo, no hay que olvidar que el desafío debe poder superarse. La satisfacción de encontrarse con una dificultad añadida para completar los objetivos requeridos puede verse truncada si dicha dificultad es excesiva, o hace la consecución de dichos objetivos algo no alcanzable. Esto normalmente hace que la satisfacción de dicho usuario sea reemplazada por una profunda frustración.

En realidad, este requisito es algo profundamente subjetivo (razón por la que se encuentra en esta sección), debido principalmente a que cada sujeto tiene sus preferencias. Hay personas que disfrutan con retos de alta dificultad, mientras que otros suelen preferir otros más sencillos. Por tanto, lo que para unos sería una experiencia de juego completa y entretenida, para otros podría llegar a convertirse en una sucesión de momentos frustrantes y poco gratificantes.

Aunque el objetivo parece más bien relacionado con el componente lúdico del juego antes que con el interés académico del proyecto, en realidad está estrechamente relacionado con este último. Una inteligencia artificial que, en una situación de igualdad, plantee cierta dificultad para el adversario, es un claro ejemplo de que se ha conseguido, a nivel general, una parte esencial de lo que se buscaba en este proyecto.

Mecánicas sencillas

Las mecánicas de juego deberían ser sencillas y fáciles de asimilar. Un juego con una gran cantidad de menús, opciones, y reglas que aprender es, por regla general, un juego tedioso y que, en muchos casos, tiene un efecto negativo en la jugabilidad del mismo. Es por tanto indispensable que se busque la sencillez, para hacer que los procesos mediante los cuales el jugador interactúa con el juego resulten lo más rápidos y naturales que sea posible.

Esto no implica que el juego en sí sea sencillo. Juegos con reglas complejas y gran variedad de opciones pueden resultar fáciles de dominar si se ofrece una serie de mecánicas adecuadas para interactuar con dichas reglas.

Entretenimiento

El juego deberá ser ameno y entretenido, a rasgos generales. No hay que olvidar que, a pesar de estar el proyecto orientado a la IA principalmente, el soporte elegido es un videojuego, y como tal

este debe de resultar interesante y ofrecer una experiencia lúdica que aumente el interés del usuario por el proyecto como conjunto.

Anticipación de la IA

La IA debe ofrecer cierta capacidad de anticipación a los movimientos del jugador, previendo algunas de las decisiones que este pueda tomar de manera más o menos acertada, y anticipándose a ellas de manera efectiva para evitar que le perjudiquen. Este es, sin duda, uno de los requisitos más ambiciosos, pues el comportamiento de los jugadores es, en muchos casos, imprevisible. En el mejor de los casos, el jugador seguirá razonamientos diferentes a los que los algoritmos de la IA, con la tecnología actual, puedan usar, lo que hace que esta capacidad de anticipación sea limitada.

Este requisito es fuertemente subjetivo, ya que debe evaluarse según el comportamiento de la IA respecto a cada jugador. ¿Ha sabido anticiparse a las decisiones de este, frustrando sus planes? ¿Considera el jugador que su enemigo era consciente de sus intenciones? Esta y otras preguntas deben de ser respondidas según la experiencia de juego o el análisis concreto del código, lo cual da lugar a discrepancias según la persona que lo analice.

Potencia de la IA

La IA debe de contar con una capacidad de procesamiento adecuada, que le permita ofrecer un desafío aceptable al jugador. Qué es aceptable en este caso, deberá decidirlo dicho jugador, pero como definición general se podría utilizar la siguiente: "Que el jugador enemigo utilice tácticas avanzadas de combate y tome decisiones ventajosas para mejorar su posición en el juego".

6.3. Especificación de requisitos

Se pasará ahora a especificar los distintos requisitos antes mencionados, orientándolos al tipo de juego que se busca y enfocándolos al resultado final.

6.3.1. Gráficos

Variedad

Para buscar variedad en el juego, cada unidad y tipo de terreno deberá tener un sprite asignado, que en ningún caso deberá ser el mismo que el de otra unidad excepto en casos puntuales y justificados. El sprite podrá ser similar en caso de unidades idénticas pero pertenecientes a bandos distintos, pero deberá estar marcado de forma correcta para indicar de forma clara a qué bando pertenece dicha unidad.

Los menús deben de contar con un aspecto diferenciado dependiendo del tipo de menú, de tal manera que el usuario sepa inmediatamente de qué se trata nada más contemplarlo.

Complejidad

Se evitarán figuras poligonales e imágenes poco descriptivas, como cuadrados o letras. Se intentará que el aspecto gráfico del juego cuente con imágenes en miniatura de cada unidad y terreno, de modo que estas estén representadas adecuadamente por versiones simbólicas de la idea a la que hacen alusión. Por ejemplo, un soldado tendría un sprite asociado mostrando un icono que haría referencia a un soldado de algún tipo, y un bosque podría mostrar otro en el que aparezcan uno o más árboles.

6.3.2. Jugabilidad

Variedad

El juego contará con diez o más tipos de unidades, cada una con sus características y especificaciones, y con cinco o más tipos de terreno. Además, cada unidad podrá realizar más de una acción, siendo estas viables o no dependiendo de las características de dicha unidad y del estado actual del juego.

Duración

Se incluirán al menos cinco niveles en el juego, de diferentes dificultades y con diferentes combinaciones de terreno y unidades. Se intentará forzar a la IA a adaptarse a distintas situaciones diferenciadas en cada nivel.

Reglas

Se introducirá un breve tutorial o un resumen de reglas en el primer nivel o en la pantalla de inicio. Además, se definirán una serie de reglas inmutables a la hora de mover, atacar y realizar otro tipo de acciones, así como unas condiciones de victoria o derrota, que afectarán tanto al jugador como a la máquina.

6.3.3. Plataforma

Instalación

La instalación del juego será sencilla, siguiendo una serie de pasos intuitivos que generen un ejecutable. En caso de haber algún proceso algo complejo en ella, se explicará de forma sencilla en un archivo adjunto.

Requisitos técnicos

Los requisitos de memoria RAM, procesador y tarjeta gráfica no serán excesivos. Se ha decidido no especificar estos datos debido a la falta de equipos en los que probar el juego, pero como valor orientativo, cualquier ordenador estándar (entendiendo por esto que tenga unas características comunes para los productos más vendidos en España en dicho año) fabricado en los últimos cinco años debería de ser capaz de ejecutar el juego sin ningún problema, siempre y cuando esté en buenas condiciones.

6.3.4. Inteligencia artificial

Herramientas teóricas

Se deberán utilizar, o al menos analizar, tres o más algoritmos o técnicas conocidas y relacionadas con la inteligencia artificial, que puedan ser aplicables a la toma de decisiones del jugador enemigo.

Entorno desconocido

El entorno de juego debe situar una o más variables o factores fuera de la capacidad de análisis de la Inteligencia Artificial, para que esta no pueda tenerlas en cuenta. Como ejemplos a evaluar, se tendrán en cuenta la capacidad de visión de las unidades (haciendo que las unidades enemigas

tengan cierta "niebla de guerra.ª la hora de analizar el terreno), cierto componente aleatorio en la resolución de los combates (que crea cierta incertidumbre) y un límite de análisis para cada unidad, perfilado en el número de aliados y enemigos que puede tener en cuenta a la hora de tomar decisiones (de modo que se controle el tiempo y el gasto de recursos).

Tiempo de espera

Como valores orientativos, se intentará evitar que el tiempo de carga de cada nivel dure más de diez segundos. Esta misma cantidad de tiempo se aplicará al tiempo que puede estar la IA ïnactiva", sin ejecutar ninguna acción. Si se supera, la IA estará obligada a elegir la mejor decisión de las que se han valorado y llevarla a cabo automáticamente, para evitar hacer esperar más al usuario. Además, se intentará evitar en todo momento que el tiempo consumido en el turno del jugador máquina sea mayor que el número de unidades que controla dicha IA, multiplicado por cinco, en segundos.

Tipos de IA

Se crearán al menos tres tipos de IA. Por ejemplo: agresiva, defensiva, neutral. La primera daría prioridad a las acciones arriesgadas, la segunda a las acciones prudentes, y la tercera sería un punto medio entre las dos primeras. Se podrá observar la diferencia entre la toma de decisiones de cada IA en una partida similar. La IA que se aplicará se decidirá en cada nivel. Si hay tiempo, se planteará una variación de la IA dinámica, en la que el jugador máquina podrá cambiar su estrategia dependiendo del estado de la partida.

6.3.5. Otros requisitos

Facilidad de análisis

Los resultados de la IA se podrán analizar, a un nivel básico, jugando un nivel.

6.3.6. Requisitos subjetivos

Sin especificación

Estos requisitos son especialmente subjetivos, por lo cual no requieren una especificación por encima de lo mencionado en la sección anterior: Gráficos atractivos, Gráficos identificables, Entorno sencillo, Mecánicas sencillas, Entretenimiento.

Desafío

El desafío debería ir en aumento, empezando por un nivel sencillo, que sirva como tutorial, y acabando en un nivel complicado que exija cierto conocimiento del juego por parte del jugador y un grado de esfuerzo adecuado para trazar los planes con los que vencer al enemigo.

Anticipación de la IA

El jugador debería ser capaz de observar como la IA se adapta a sus planes más sencillos de forma efectiva.

Potencia de la IA

Un jugador novato debería encontrar una dificultad moderada al enfrentarse contra un ejército de unidades enemigas de potencia similar al que dicho jugador controla.

Capítulo 7

Entorno

7.1. Buscando un entorno de trabajo

Para realizar el proyecto propuesto, se necesita un entorno de diseño y programación que nos permita completar nuestros objetivos con facilidad. Al ponerse a comparar las distintas opciones disponibles para el diseño de videojuegos que circulan por internet, se encuentran algunas alternativas interesantes que nos ofrecen distintas posibilidades a la hora de construir el proyecto que se tiene en mente. Resulta pues, indispensable, realizar un breve trabajo de comparativa que permita elegir entre uno u otro, pues dar marcha atrás en mitad de la implementación resultaría excesivamente costoso.

Para guiarse por parámetros relativamente objetivos, es importante fijarse en las características que se buscan para hacer del proceso de diseño, planificación e implementación del programa lo más dinámico, fácil y cómodo posible. Estas características se pueden dividir en varios campos, y aunque es necesario imponer una serie de criterios arbitrarios para evaluarlas, y establecer un peso para cada una de ellas en la toma final de decisiones, se intenta también que estas puedan ofrecer un análisis realista de lo que ofrece cada entorno.

Sin embargo, es conveniente recordar que el principal objetivo del proyecto no es la evaluación exhaustiva de estos entornos, que más bien son una herramienta. El tiempo, pues, para realizar dicha evaluación no será tan amplio como el de un proyecto específicamente dedicado a ello. Esta limitación de tiempo y recursos supone que al establecer los criterios antes mencionados estos sean necesariamente simples y sencillos de evaluar, para analizar cada opción de manera rápida y cómoda.

1. El entorno de trabajo debe ofrecer un **soporte gráfico** que facilite la carga de sprites y diseño de niveles. Se quiere centrar el esfuerzo en el desarrollo de la IA, siendo pues deseable encontrar una serie de herramientas que permitan reducir al máximo la carga de trabajo en lo que respecta a la parte más visual del juego.

2. El entorno de trabajo debe contar con una serie de herramientas para simplificar al máximo la implementación del esqueleto del juego, como podría ser la carga y transición entre niveles o las físicas de los objetos, para poder centrar la mayor parte del esfuerzo en el diseño de la IA.

3. El entorno de trabajo debe ofrecer un sistema de eventos, comportamientos o programación en profundidad que permita realizar un **sistema de IA** adecuado a lo que se está buscando.

4. El entorno de trabajo debe ser **gratuito**, o tener un coste asequible.

Siguiendo estos tres objetivos, se puede deducir que, en resumen, el entorno más adecuado será aquel que se centre en un enfoque de programación a alto nivel, que permita modificar los gráficos de forma simple y rápida, y que permita empezar con el desarrollo de la IA lo más rápido posible. Con estos parámetros en mente, se puede dividir algunos de los entornos de desarrollo de videojuegos que circulan por el mercado actualmente en varias categorías.

Nótese que hay una gran cantidad de entornos que se han dejado fuera de este análisis debido a varias razones, como la falta de tiempo o la poca influencia que tienen en el mundo del desarrollo de videojuegos. Otros tantos entornos han sido directamente descartados debido a su uso exclusivo en ámbitos profesionales o a su alto precio en el mercado.

7.2. Entornos para profesionales

Normalmente, al pensar en herramientas gratuitas de desarrollo de videojuegos, lo primero que viene a la mente son videojuegos 'casuales', asociados con el género *indie*, o pequeños proyectos sin ánimo de lucro orquestados por desarrolladores *amateur*. Sin embargo, entre los distintos entornos que podemos encontrar circulando entre el mercado, algunos ofrecen verdaderas posibilidades para la creación de videojuegos a un nivel competente y profesional.

Desde el 'HearthStone: Heroes of Warcraft' o el 'Assasin's Creed: Identity' del motor Unity [6] hasta los inicialmente menos ambiciosos 'GunPoint' o 'Hotline Miami' [16], cada vez más juegos de corte profesional creados con motores gratuitos o asequibles para la gran mayoría de desarrolladores independientes se van haciendo un hueco en el mercado global. Aunque los ingresos que estos juegos producen para sus creadores no se acercan al negocio 'de alto nivel' como el de las grandes compañías, sí que allanan el camino para que dichos desarrolladores independientes, o sus pequeñas empresas, se hagan un pequeño nombre entre la comunidad gamer y en el mundo de los juegos 'indie'.

Por tanto, es inevitable que, con el paso de los años, algunos de los motores de diseño de videojuegos en este ámbito se hayan ido convirtiendo en herramientas más profesionales, detalladas y cuidadas, capaces de ofrecer grandes prestaciones a todo aquel que aprenda a dominarlas y dedique suficiente tiempo y esfuerzo a trabajar con ellas. Juegos cada vez más competentes nacen de pequeños estudios conformados por pequeños grupos de diseñadores. Y no hay que mirar muy lejos para encontrar motores como el UDK, versión libre del Unreal Engine 3, responsable de videojuegos tan famosos como 'Gears of War', 'Tom Clancy's Rainbow Six: Vegas' o 'Mass Effect', que hace algo más de cinco años fue liberado para su uso por la comunidad de programadores.

Con tantas opciones, no resulta extraño observar que la comunidad está dividida a la hora de decidir qué motor resulta más adecuado, potente o fácil de utilizar. Como siempre, las modas juegan un importante papel en esta división, y aunque hace unos años la popularidad de algunos productos podía ser bastante grande, hoy en día su número de seguidores se ha visto drásticamente reducido a favor de otras opciones más atrayentes.

Pero esta no es, lógicamente, la única razón de que exista un significativo número de entornos disponibles. Es fácil darse cuenta de que las necesidades de un *shooter* en tres dimensiones serán radicalmente distintas a las de un *rpg* por turnos en dos dimensiones. Ante un abanico inmenso de posibilidades y una serie de marcadas diferencias entre géneros opuestos, diseñar un motor capaz de ofrecer todas estas posibilidades al mismo tiempo entra en conflicto necesariamente con la posibilidad de hacer que ese mismo motor trabaje a un alto nivel. Aquellos motores que han intentado convertirse en una 'herramienta para todo' no siempre han fallado en su propósito, pero la curva de aprendizaje y la cantidad de trabajo requerido son en muchas ocasiones significativamente mayores a las de otros motores más especializados en un tipo u otro de juego.

Surgidos de este pequeño dilema se perfilan algunos motores orientados específicamente a un tipo

concreto de juegos, que en los últimos años han ido ganando cada vez más popularidad. Opciones especialmente cerradas como RPG Maker o Adventure Game Studio permiten realizar con gran sencillez y asombrosa fluidez proyectos de juegos englobados en géneros de rol o aventura gráfica, respectivamente. Game Maker se perfila como una de las opciones preferidas para juegos en dos dimensiones, mientras que UDK es sin duda uno de los motores más potentes para juegos de tres dimensiones. Dependiendo de lo que el desarrollador busque, tiene a su mano una gran variedad de herramientas entre las que elegir la que más se adapte al proyecto que tiene en mente.

Sin embargo, en algunos casos se ha podido diseñar un producto que satisface a gran parte de la comunidad de desarrolladores, y que es capaz de ofrecer una amplia variedad de herramientas con las que crear distintos tipos de videojuegos. Uno de los mejores ejemplos es Unity, que aún hoy en día sigue siendo uno de los motores más populares y utilizados por los usuarios. Algunos programadores, incluso, prefieren abordar sus proyectos de forma directa, programando sobre lenguajes de alto nivel como C++.

Para analizar cada entorno se resumirá su historia, sus principales virtudes y defectos, y se compararán todas estas características con los tres requisitos que se habían propuesto en el punto anterior.

7.3. Análisis de entornos

7.3.1. Unity

Es inevitable empezar hablando de uno de los motores más populares del mercado actual, usado en varios juegos comercialmente exitosos como pueden ser el estratégico Wastelands 2, el deslumbrante Ori and the Blind Forest o el curioso Kerbal Space Program. Usado tanto por grandes empresas como por pequeños grupos de desarrolladores, es la primera opción de muchos, e incluso aquellos que prefieren otros motores más potentes como Unreal Engine 4 o CryENGINE suelen realizar críticas positivas ante la versatilidad y fácil manejo de Unity.

Ideado en los primeros años del cambio de milenio por tres programadores aficionados [3], sería uno de los primeros motores en ofrecer distintas funcionalidades para el desarrollo de juegos, sobre todo relacionadas con las físicas y el manejo de los gráficos. Aunque en principio orientado a juegos de móvil y ordenador, las últimas actualizaciones lo convierten en una opción cada vez más viable para diseñar juegos para consolas de última generación y otros soportes poco frecuentes.

Aunque algunas grandes compañías se han interesado en Unity y lo han utilizado para realizar diferentes videojuegos en el pasado, los responsables del motor han afirmado en varias ocasiones que prefieren centrarse en hacer de su producto algo asequible para la gran mayoría de desarrolladores amateur y pequeñas empresas en alza. Es quizás por eso por lo que Unity prefiere sacrificar algo de la potencia gráfica de algunos de sus competidores a cambio de una interfaz sencilla y una gran cantidad de herramientas disponibles para el usuario.

Carece de un sistema de manipulación de gráficos y texturas similar al de algunos de sus rivales, pero su compatibilidad con diversos tipos de formatos hace que pueda funcionar bien en combinación con software de animación como Maya, 3ds Max, Softimage y muchos otros. También trabaja bien con sprites y físicas 2D, haciéndolo una buena opción de cara al diseño de juegos en dos dimensiones.

Algunas de sus principales ventajas son:

1. Entorno relativamente sencillo.
2. Gran variedad de opciones de desarrollo.
3. Comunidad de usuarios dedicada y amplia.

4. Posibilidad de trabajar con C# o Javascript.

5. Capaz de trabajar fluidamente con buenos gráficos en 3D.

En lo que respecta a los requisitos que se habían decidido, el entorno parece poder adaptarse correctamente a lo que se busca. Al ser un entorno versátil, es posible que la carga de trabajo sea mayor frente a la de otros más centrados en lo que se busca específicamente. El soporte gráfico es adecuado, permitiendo realizar la implementación básica del juego y el sistema general con sencillez. Al tener una versión gratuita, queda como una muy buena opción a tener en cuenta.

Figura 7.1: Unity se perfila como uno de los motores gratuitos más populares

7.3.2. Unreal Development Kit

Aunque ahora eclipsado por su versión más reciente, Unreal Engine 4, durante muchos años UDK se ha perfilado como una de las opciones más asequibles para aquellos desarrolladores sin muchos recursos económicos pero que buscaban realizar productos complejos y visualmente equiparables a algunos de los de las grandes compañías sin tener por ello que invertir grandes cantidades de dinero y personal en ello. Algunos proyectos de UDK distaban mucho de parecerse a lo acostumbrado hace unos años en lo referente a juegos de entidades independientes. Más de un jugador probaría esos juegos y notaría poca diferencia respecto a otros creados por las grandes marcas del mercado. Con una potencia que solo ponía trabas al esfuerzo, talento y número de personas que pudieran dedicarse al proyecto, UDK no tardaría en convertirse en uno de los grandes nombres de la industria.

No hay más que ver la lista de juegos creados con UDK para darse cuenta de su relevancia en los últimos años [24]. Asura's Wrath, Army of Two, Batman: Arkham City, Bioshock Infinite, Borderlands, Gears of War... La lista continúa y continúa. Y, como se ha mencionado antes, también con juegos independientes. Su sucesor, Unreal Engine 4, que ha creado gran expectación desde su total liberación (exceptuando el porcentaje que recibe Epic de los beneficios de los juegos creados

con el motor) para la comunidad de desarrolladores en marzo del 2015, ya se ha agenciado títulos como Kingdom Hearts III, Dead Island 2 o Fable Legends, así como otros proyectos igualmente prometedores.

Una de las más aclamadas características de este motor es el lenguaje de scripts que utiliza: Unrealscript. Considerado una manera muy sencilla de añadir comportamientos e introducir el propio código en el juego, ha sido sin embargo reemplazado en su sucesor por una opción mucho más conocida y flexible: C++. La calidad de los gráficos es otro de sus más reconocidas virtudes, mientras que la curva de aprendizaje es considerada rápida y amistosa para nuevos desarrolladores. En la otra cara de la moneda, el número de soportes al que se pueden exportar los productos finales creados con el motor es relativamente reducido en comparación a otros motores como Unity.

Unreal Engine 4 ha añadido varias mejoras que son merecedoras de mención, como la implementación de las 'Blueprints', un sistema de scripting que no exige al diseñador escribir ninguna línea de código, lo que lo hace especialmente útil para aquellos nuevos desarrolladores que no tengan demasiados conocimientos de programación [15]. Para los que sí los tengas, el uso de C++ permite gran cantidad de opciones, que un programador experimentado podrá usar para plasmar sus ideas en el juego sin un esfuerzo excesivo. Por último, la mejora gráfica es notable, tanto que actualmente está considerado en muchos lugares como la mejor opción *indie* para hacer videojuegos de calidad, puesto que se disputa con otras opciones igualmente conocidas como CryEngine [14].

Algunas de sus principales ventajas son:

1. Potencia gráfica considerable.
2. Sistema de scripting fluido.
3. Comunidad de usuarios dedicada y amplia.
4. Posibilidad de trabajar con C++ (en el Unreal Engine 4).
5. Permite crear videojuegos de gran calidad.

El entorno cumple en cierta medida con los requisitos. Aunque está principalmente orientado a juegos 3D, también puede adaptarse de forma exitosa a juegos 2D. Sin embargo, el hecho de que esté orientado principalmente a juegos de mayor calidad del que se pretende en este proyecto, es bastante probable que la carga de trabajo sea mayor que la de otros entornos orientados a las dos dimensiones. Al ser gratuito (y orientar el proyecto a un uso principalmente académico y no comercial), queda como una opción viable, pero quizás no la más adecuada debido a que su excesiva potencia puede ser contraproducente al buscarse algo más simple.

Figura 7.2: El repertorio de juegos creados con UDK incluye una gran cantidad de títulos famosos

7.3.3. CryEngine

Como una de las principales alternativas a Unreal Engine 4, CryEngine, ideado por CryTek, lleva desde su primera iteración demostrando que bien puede estar a la altura de su contraparte con juegos como FarCry, Crysis, Ryse: Son of Rome o Evolve. Ideado inicialmente como una demo para NVidia, CryTek no tardó en convertirlo en un popular motor al ver su potencial. Actualmente se encuentra en su tercera versión, aunque CryTek ha decidido cambiar el nombre de CryEngine 3 a CryEngine debido a que su siguiente proyecto, según han confirmado, tendrá un nombre distinto para resaltar las diferencias entre ambos.

CryEngine ofrece, como una de sus principales características, la posibilidad de usar su editor sandbox para diseñar niveles y entornos multiplataforma sin necesidad de invertir tiempo en adaptarlos específicamente a cada una. Su filosofía es 'WYSIWYP': What You See Is What You Play. Siguiendo esta idea, el motor permite editar animaciones en tiempo real, y ofrece a los diseñadores todo tipo de herramientas para hacer del desarrollo una experiencia lo más visual y directa posible.

Orientado también a juegos 'sandbox', con terrenos kilométricos que el jugador puede explorar, se centra en herramientas de renderizado para mundo abierto: Niveles grandes que requieren diferentes tipos de visualización dependiendo de lo cerca que se encuentre el jugador de según qué elementos. Su capacidad gráfica, en general, es la parte más valorada del motor, con unos resultados bastante realistas en comparación a lo acostumbrado en los contemporáneos del género, aunque algunos de sus críticos afirman que la dificultad de aprendizaje es mayor que la de su contraparte, el Unreal Engine 4, que muchos consideran un entorno más amistoso [19].

La comunidad de CryEngine es considerablemente más reducida que la de Unreal Engine. Esto es principalmente debido a que, pese a los diferentes gustos de cada desarrollador, el segundo ha servido para crear una mayor cantidad de títulos a fecha de hoy, y por tanto es relativamente más popular. Sin embargo, algunos afirman que CryEngine solo ha empezado su trayectoria, y que su potencial es suficiente para convertirlo en uno de los principales motores de los próximos años.

Algunas de sus principales ventajas son:

1. De lo mejor en cuanto a gráficos en relación a sus competidores.
2. Facilita el diseño de la Inteligencia Artificial.
3. Editor 'sandbox', visual e intuitivo.
4. Lenguaje de scripting Lua.
5. Permite crear videojuegos de gran calidad.

Como con el Unreal Engine, comprobamos que este entorno también cumple algunos de los requisitos. Sin embargo, a diferencia de este, CryEngine no está tan orientado a juegos en dos dimensiones, y la mayor parte de productos creados con este motor son en tres dimensiones. De hecho, la mayor parte son FPS, o *First Person Shooter*. La carga de trabajo, previsiblemente, será grande en comparación a otros motores orientados a juegos más sencillos, como el que se plantea hacer. El hecho de que requiera un suscripción mensual de 9.90 euros no supone un gran coste, por otro lado, en comparación a las prestaciones que ofrece.

Figura 7.3: A nivel profesional, CryEngine se perfila como una de los mayores competidores de UDK

7.3.4. Microsoft XNA

Aunque a día de hoy se encuentra relativamente abandonado, XNA fue una popular propuesta de Microsoft para trabajar en la plataforma .NET, orientada sobre todo a juegos 'indies' frente a su más popular sistema dentro de la misma compañía, DirectX. Durante varios años, el Framework de XNA se convirtió en una conocida opción para desarrolladores independientes y pequeñas empresas, que si bien no ofrecía un entorno visual tan sencillo como otras propuestas, sí que permitía una programación eficiente para aquellos versados en C#.

XNA es gratis, a menos que se deseen publicar juegos en Xbox Live, y ofrece una herramienta muy útil para el desarrollador que decide elegirlo como entorno: XNA Studio. Actualmente en su cuarta iteración, facilita enormemente la depuración del código y la reutilización de librerías y código orientado a videojuegos. A pesar de que no ofrece una herramienta de diseño de niveles similar a la de otros entornos, sí que cuenta con algunas extensiones que permiten suplir esta necesidad, en muchos casos desarrollados por los propios usuarios de la comunidad.

En abril del 2014, Microsoft anunció que dejaría de apoyar el proyecto, y no hay prevista una nueva versión de XNA Studio, por lo que la comunidad que apoyaba este entorno se ha ido disolviendo poco a poco. Actualmente, y aunque todavía hay algunos juegos que utilizan la tecnología previstos para el 2015, el número de desarrolladores que eligen esta opción es bastante escaso en comparación a otros entornos. Aunque proyectos como MonoGame [17] parecen estar intentando rescatar el entorno, su futuro es incierto.[9]

Algunos de los juegos más conocidos realizados con XNA son Reus, Schizoid, Dishwasher: Dead Samurai o Rogue Legacy. Aunque la mayoría de los juegos creados con este entorno son juegos en dos dimensiones, XNA permite la creación de juegos en tres dimensiones, si bien se recomienda usar alguna de las extensiones aportadas por los usuarios de la comunidad o programar un motor propio para poder facilitar la tarea.

Algunas de sus principales ventajas son:

1. Manejo de recursos con XNA Build, que permite eliminar elementos no utilizados.
2. Permite trabajar con C#.

Este entorno, en principio, no cumple con los requisitos que se habían propuesto. Aunque permite una programación sencilla y rápida, requiere una cantidad de trabajo adicional para el diseño de niveles y manejo de gráficos frente a otros entornos más 'amigables' en ese aspecto.

Figura 7.4: El futuro de XNA es incierto

7.3.5. Torque 2D

Torque lleva en el mercado más de un lustro, y aún hoy en día sigue siendo la opción preferida de muchos desarrolladores para sus creaciones. Aunque durante unos años ha sido desplazado en cierta medida por otros entornos más amigables, potentes o adaptados a las nuevas tecnologías, su historial lo perfila como uno de los grandes nombres en el mundo *indie*. Aunque no tan popular como otros motores como Unity o Game Maker, ha sido utilizado en títulos conocidos como Blockland, Minions of Mirth, S.P.A.Z. o Frozen Synaspe.

Torque cuenta con una historia turbulenta. Aunque empezó como un entorno relativamente popular en sus primeras versiones, poco después de ser adquirido por la compañía IstantAction comenzó a recibir varias críticas sobre su falta de documentacion, numerosos *bugs* y mala compatibilidad con algunas de las nuevas tecnologías. Este y otros factores hicieron que su popularidad decreciera hasta que, en el 2011, fue adquirido de nuevo por GarageGames, sus primeros dueños. Con un nuevo equipo a manos del producto, Torque ha vuelto a crecer en seguidores, y aunque no resulta una opción tan popular como algunos de los grandes títulos de hoy en día, no son pocos los que apuestan y prevén un futuro prometedor para el motor.

Lo más característico de Torque, a día de hoy, es sin duda la influencia de la comunidad en el propio entorno: En septiembre del 2012 fue liberado con licencia MIT, esto es, libre, gratis y, lo que es más importante, abierto a toda la comunidad para observar, analizar e incluso modificar el código. Usando esta filosofía *open source*, Torque se ha convertido en uno de los motores que más comunicación y compenetración ofrece entre la comunidad de usuarios y los propios desarrolladores del motor. Estos últimos han llegado a manifestar su intención de mantener un estrecho contacto con los usuarios del motor, resolviendo dudas y desarrollando documentación de forma constante para facilitar el aprendizaje y uso de este[22].

Actualmente, Torque se divide en dos grandes derivados: Torque 2D y Torque 3D. Como se puede deducir por sus nombres, cada uno se especializa en realizar un tipo concreto de juegos, dependiendo de si estos se encuentran en dos o tres dimensiones. Con el tiempo los dos se han ido separando hasta convertirse en dos productos prácticamente independientes. Mientras que Torque 3D contiene un editor del estilo *WYSIWYG*, Torque 2D utiliza una GUI algo más sobria y centrada en el código, que parte de una consola en la cual se introduce el código deseado, y hasta la fecha todavía no cuenta con ningún editor oficial.

Algunas de sus principales ventajas son:

1. Comunidad activa y contacto con los desarrolladores.

2. Posibilidad de acceder a todo el código del motor.

3. Posibilidad de trabajar con C++.

Aunque la licencia MIT y la filosofía subyacente resultan atractivas, la falta de un editor sencillo y la necesidad de trabajar constantemente con el código no parecen comulgar con los requisitos establecidos, en especial con la idea de obtener un entorno que facilite al máximo el diseño general del juego y permita centrarse en la IA.

Figura 7.5: La licencia MIT es sin duda una de las decisiones más importantes de Torque

7.3.6. Game Maker

Originalmente presentado como Animo, Game Maker es uno de los motores más utilizados en el panorama actual en lo que se refiere a juegos casuales en 2D. Usado por profesionales y programadores amateur por igual, se ha usado para crear videojuegos tan dispares (y, en muchos casos, famosos), como Desktop Dungeons, Gunpoint, Hotline Miami, Stealth Bastard o Wanderlust Adventures, entre otros. Orientado de forma exclusiva a gráficos 2D, su principal mercado es el de los juegos *indie*, en donde se ha consagrado como una de las opciones más recomendadas para iniciarse en el desarrollo de videojuegos.

En la actualidad, el producto principal de YoYo Games, la empresa a la que pertenece el motor, es Game Maker Studio. Este entorno ofrece un editor visual sencillo e intuitivo, que sigue la filosofía WYSIWYG. Dicho editor facilita la creación de niveles, y ofrece una serie de físicas y eventos que facilitan la creación de juegos de plataformas o acción en dos dimensiones. Game Maker ofrece la posibilidad de diseñar el juego usando una interfaz *drag and drop*, basándose en un sistema visual de eventos y disparadores para aquellos con pocos conocimientos de programación, o un sistema de scripts para aquellos con más conocimientos. Mezclando los dos, ofrece cierta versatilidad dependiendo de los conocimientos que se tengan sobre programación para poder crear un juego rápidamente sin renunciar a poder modificar aspectos sobre el funcionamiento interno de este más adelante.[18]

Game Maker usa una popular librería de físicas: Box2D. Esta librería allana el camino a la hora de crear videojuegos de algunos géneros populares en el mercado, como las plataformas o la acción, y ahorra tiempo al poder alterar los parámetros para que se ajusten a las necesidades del diseñador en vez de obligar a este a programar dichas físicas desde cero. El sistema de scripting que utiliza el entorno es conocido como GML (Game Maker Language), que aunque simple resulta efectivo para trabajar con algunos de los aspectos más internos del juego como la IA o algunos comportamientos complejos de los objetos. Por último, también ofrece varias conexiones con los servicios del mercado para publicar, promocionar y monetizar los juegos creados con el entorno.[4]

Aunque Game Maker cuenta con una versión gratis del motor para aquellos que quieran probarlo o usarlo para crear juegos no comerciales. Ofrece también una versión de pago, sin limitaciones de recursos y con menos restricciones a la hora de comercializar los juegos creados usándolo. Sin embargo, para exportar a diferentes plataformas se han de comprar extensiones adicionales por cada una. Game Maker ha levantado ciertas polémicas en lo que respecta a los derechos de autor, debido a una serie de errores que aplicaban políticas anti-piratería a usuarios que habían obtenido el producto de forma legal.

Algunas de sus principales ventajas son:

1. Entorno sencillo y fácil de utilizar.

2. Sistema de físicas Box2D.

3. Gran número de tutoriales y documentación.

4. Facilidad de trabajo con sprites y gráficos 2D.

Ofrece, en conclusión, un entorno sencillo para crear el esqueleto del juego, los niveles y las reglas básicas. Aunque su sistema de físicas no será necesario (no es un juego de plataformas) es de esperar que las funcionalidades que ofrece hagan sencilla la tarea de mover las unidades y objetos por el mapa con sencillez. Un aspecto negativo es las limitaciones de GML frente a otros lenguajes usados por diferentes motores, como C#. En cuanto al precio, hay una versión gratuita, que podría ampliarse por un precio asequible en caso de que las limitaciones de recursos impidieran realizar el proyecto deseado.

Figura 7.6: Game Maker es uno de los motores que más facilidades ofrece a la hora de crear videojuegos en 2D

7.3.7. Construct 2

HTML5 ofrece muchas posibilidades, y Construct 2 es el claro ejemplo del interés actual por dicha tecnología. Enfocado principalmente a desarrolladores de juegos alternativos, y ofreciendo un aspecto sencillo y unas herramientas fáciles de utilizar, nació como un proyecto común de un grupo de estudiantes en el 2007, con el nombre de *Classic Construct*. Actualmente se encuentra en su segunda iteración, si bien en el blog de Scirra ya se ha anunciado la siguiente versión, Construct 3, que al parecer incluirá nuevas opciones de compatibilidad y soporte [10].

Si hay que destacar algún aspecto esencial de Construct 2, este sería probablemente lo intuitivo de su manejo. El entorno se centra principalmente en la filosofía *drag and drop*: Toda la programación y el diseño de los niveles se realiza de forma visual, arrastrando las imágenes y objetos o eligiendo entre varios tipos de eventos, atributos y comportamientos para cada elemento del juego. El sistema resulta intuitivo, y una vez aprendido es sencillo de utilizar, pudiendo desarrollar nuevos juegos en cuestión de horas. Esta, sin embargo, es también una de sus principales carencias, ya que al buscar la sencillez y no permitir al desarrollador experto introducir sus propias líneas de código, puede resultar algo tedioso si se quieren implementar métodos complejos como, por ejemplo, una Inteligencia Artificial avanzada para los enemigos.[20]

Sin embargo, el entorno no se presenta como una herramienta para la creación de videojuegos complejos, sino como un acercamiento a los desarrolladores sin experiencia en la programación. La mayoría de los juegos creados con este entorno son, por tanto, de tono desenfadado y orientados a géneros tradicionales como el arcade o los puzzles, si bien las posibilidades que ofrece no se limitan a estos géneros, pudiendo crear con algo de esfuerzo RPGs o aventuras gráficas, entre otros. Buenos ejemplos de lo que se puede conseguir son "The Next Penelope", Çosmochoria.º "Mortar Melon". Es también una herramienta perfecta para crear prototipos o para probar ideas que luego llevar a cabo en otro entorno más complejo.

Es necesario señalar algunas limitaciones obvias a la hora de trabajar con el entorno. La principal restricción es la imposibilidad de diseñar juegos en tres dimensiones, al menos sin aplicar modificaciones críticas o invertir una gran cantidad de tiempo en implementarlas. También se critica su rendimiento, que todavía se ve limitado por la novedad de HTML5 y que, en algunos casos, puede suponer algunos comportamientos inesperados. Por último, los videojuegos creados con Construct 2 están limitados a ciertas plataformas, y su funcionamiento en sistemas móviles como Android o iOS resulta algo aparatoso si no se integra con cuidado [13]. El entorno actualmente cuenta con una versión gratuita que, por otro lado, guarda bastantes limitaciones respecto a la versión estándar, de pago.

Algunas de sus principales ventajas son:

1. Entorno sencillo y rápido de utilizar.

2. Comunidad dedicada y gran variedad de plugins.

En lo que respecta al proyecto que se está buscando, el entorno no parece ser adecuado. Puede que simplifique enormemente la parte gráfica, pero al no ofrecer grandes opciones para el análisis avanzado del entorno o la toma de decisiones complejas, ni permitir la programación con código, limita en gran medida las posibilidades de la inteligencia artificial, que es a grandes rasgos lo que se está buscando. En cuanto al precio, la versión gratuita está algo limitada, pero podría ampliarse de ser necesario por un precio considerable aunque moderado.

Figura 7.7: Construct2 es uno de los entornos más sencillos de utilizar actualmente en el mercado

7.3.8. Otros entornos

En el mercado actual hay decenas de entornos de desarrollo de videojuegos, algunos orientados específicamente a un género, otros más generales. Aunque no se han analizado tan en profundidad como los antes citados, los dos siguientes entornos han sido estudiados y descartados por diversas razones:

1. **Stencyl**: Se puede posicionar sin mucho error entre Game Maker y Construct, siendo su complejidad algo intermedio entre esos dos entornos. No parece ofrecer muchas posibilidades para la Inteligencia Artificial que se está buscando.

2. **Shiva 3D**: Es un motor potente, al nivel de Unity, pero carece de una comunidad tan dedicada y usa una manera de acercarse al desarrollo algo peculiar. Se ha descartado debido a su complejidad.

7.4. Comparando entornos

Una vez sopesados los pros y los contras de estos entornos, se debe considerar cual cumple mejor los requisitos antes enumerados:

7.4.1. Soporte Gráfico

Para buscar la mayor sencillez y rapidez en cuanto a la carga de sprites y el diseño de los gráficos del juego, es lógico pensar que los entornos orientados al 2D harán mejor el trabajo. La opción lógica parece Construct 2, aunque Game Maker Studio o Unity podrían ser buenas alternativas. Los tres

entornos ofrecen una tecnología sencilla a la hora de arrastrar las imágenes y crear los distintos objetos, pudiendo observar el resultado de forma inmediata sin necesidad de cargar el juego.

7.4.2. Implementación del esqueleto

La mayoría de los entornos podrían cumplir esta definición, aunque los más versátiles como Unity o UDK podrían requerir una mayor inversión de tiempo a la hora de establecer las directrices necesarias para que el juego vaya tomando forma. De nuevo, Game Maker y Construct se perfilan como las opciones más sencillas.

7.4.3. Facilidad de implementación de IA

Se puede descartar automáticamente Construct 2, que debido a su simplicidad seguramente requerirá un trabajo adicional considerable para el desarrollo de la IA del bando enemigo. Las mejores opciones parecen Unity o Torque 2D, que permiten trabajar respectivamente con C# y C++, dos lenguajes de alto nivel que ofrecerían muchas posibilidades de programación.

7.4.4. Precio

Ninguno de los motores estudiados parece excesivamente caro en sus versiones más baratas.

7.5. Eligiendo el motor

Después de la comparación, cabe recalcar que hay varios entornos que cumplen con los requisitos de forma adecuada. Unity y Game Maker se perfilan como las opciones más prometedoras, no solo por su popularidad y las prestaciones que ofrecen, sino por las características objetivas de dichos entornos.

Entre los dos, se ha decidido usar Game Maker, sacrificando un lenguaje más potente a cambio de una mayor facilidad para tratar los gráficos. Esta decisión ha sido arbitraria, basándose en las preferencias personales debido al aspecto y a las críticas leídas.

7.6. Evaluación final

Tras utilizar Game Maker, se pueden extraer algunas conclusiones de la propia experiencia a la hora de programar con el motor. De esta manera, se evaluará si la decisión al utilizar Game Maker ha sido óptima, o si la experiencia podría haberse mejorado de haber usado otro motor de los mencionados.

Algunos de los aspectos más destacados han sido los siguientes.

7.6.1. Diseño de niveles

El diseño de niveles del juego ha resultado, como era de prever, relativamente sencillo y cómodo. Game Maker se basa en un sistema de rooms", siendo cada room un nivel distinto del juego que se carga automáticamente siguiendo un orden establecido por el usuario. Se pueden realizar transiciones entre las rooms de forma sencilla y rápida, pasando directamente a la siguiente en el orden establecido o saltando de una a otra con rapidez. Esto quiere decir que se puede fácilmente diseñar cada "zona"del

juego por separado y ponerlos en orden para establecer una serie de pantallas por las que el juego se moverá, incluyendo el menú de inicio o los distintos niveles de juego.

Se ha optado por crear una room de carga que se limite a utilizar un fichero para extraer la información de cada nivel, descargando luego dichos datos en pantalla para empezar el juego. Asimismo, se ha diseñado un nivel de guardado, en el que mediante un sistema de tiles y objetos ya definidos se puede crear un nuevo nivel desde cero, que el código luego interpreta y analiza para generar un fichero compatible con el código de carga situado en la room antes mencionada.

El entorno ha ofrecido muchas facilidades a la hora de realizar este trabajo inicial, y el diseño de niveles es sencillo e intuitivo, pudiendo crear nuevas dificultades para el usuario sin demasiado esfuerzo. Por tanto, es fácil considerar que este aspecto ha supuesto una ventaja reseñable en los primeros pasos del diseño del programa, una ventaja con la que posiblemente no se habría contado de haber trabajado con otros entornos.

7.6.2. Físicas

Game Maker ofrece un montón de opciones para juegos con fuerte influencia de físicas, como los juegos de acción o plataformas. Detalles como la gravedad o las colisiones son soportados y facilitados de una forma asombrosa. El motor permite resumir unos comportamientos sencillos en una serie de opciones que se pueden aplicar a los distintos objetos del juego marcando una serie de opciones en apenas unos segundos. Comportamientos que, de otro modo, llevaría tiempo y esfuerzo programar.

Sin embargo, la influencia de las físicas en el juego que se estaba planeando era, como poco, baja. Al ser un tablero, las piezas tienen una serie de movimientos establecidos de antemano, y aspectos como la velocidad o la inercia son secundarios. Al ser Game Maker un motor tan fuertemente orientado a estas físicas antes mencionadas, se ha preferido utilizar estos sistemas de colisiones y movimientos con una serie de modificaciones que permitan simular dichas jugadas en el tablero de juego.

El resultado es adecuado, y las opciones que ofrece el entorno han permitido realizar este esqueleto de juego sin una cantidad excesiva de trabajo, pero se plantea la idea de que en otro de los entornos analizados, menos orientado a este tipo de físicas, estos movimientos y distribuciones pudieran haber sido aún más sencillo.

7.6.3. Programación

A la hora de programar los distintos scripts necesarios para el desarrollo del juego, así como la IA, se ha descubierto que el lenguaje que ofrece Game Maker, GML, es bastante menos potente de lo que se había planteado en un primer momento. No solo no ofrece facilidades a la hora de aplicar el paradigma de programación orientada a objetos, sino que en muchos casos, lo dificulta. Esto es debido principalmente a que el único aspecto de GML que puede usarse como un objeto en este paradigma, es justamente los objetos que se muestran en cada pantalla. Esto no sería un problema si no fuera por la cantidad de recursos que consume cada uno de estos objetos, y la dificultad de encontrarlos una vez fijados en la room. Todavía se puede usar este sistema para crear objetos invisibles que realicen diversas funciones sin estar directamente relacionados con aquellos que sí tienen una representación real en pantalla, pero se obtiene la sensación de que se está usando una solución más bien chapucera y poco cuidada, y la falta de un explorador adecuado hace que la búsqueda de dichos objetos, una vez creados, sea tediosa.

Otro problema es la falta de funciones y estructuras básicas, presentes en otros sistemas. Fue necesaria la implementación de una serie de funciones para tratar con los arrays, tan básicas como la introducción de elementos o la creación de nuevos vectores. Este trabajo, aunque sencillo, implica un

gasto de tiempo y trabajo que sería innecesario de contar el lenguaje con funciones más avanzadas. Sin embargo, la mayoría de las funciones están orientadas a la interacción con los distintos elementos de cada room, y no al procesamiento interno.

Por desgracia, es esta capacidad de procesamiento interno y simplicidad del flujo de trabajo lo que más se necesitaba en lo que respecta a la Inteligencia Artificial, y aunque el análisis de elementos del terreno fue relativamente sencillo, la parte posterior de toma de decisiones resultó larga y poco estructurada. Esta ha sido, sin duda, la principal debilidad de Game Maker a la hora de realizar este proyecto.

7.7. Conclusiones

Aunque el grado de satisfacción al haber usado este motor es adecuado, se plantea que, de haber usado Unity, el montaje de la IA (en C#) habría sido más cómodo. Puede que Game Maker haya ofrecido una serie de facilidades a la hora de trabajar con los gráficos, pero es planteable la posibilidad de que al haber sacrificado estas facilidades se hubiera obtenido una mayor potencia para diseñar el comportamiento del jugador enemigo, que, al fin y al cabo, era el objetivo principal del proyecto.

Capítulo 8

El juego

8.1. Nociones básicas

Para empezar a definir el juego, se deben tener en cuenta distintas nociones que no son comunes a todos los juegos de estrategia. Estos factores se han decidido teniendo en cuenta los requisitos y las preferencias personales a la hora de diseñar el juego:

1. El juego es para **un solo jugador**. Este compite contra la máquina en distintos niveles.

2. El juego se desarrolla en distintos **turnos**. En cada uno de sus turnos, el jugador tiene un tiempo ilimitado para decidir su estrategia y dar órdenes a sus unidades.

3. Las decisiones son **irreversibles**. Una vez se da una orden a una unidad, esta realiza dicha acción y se generan los resultados antes de que el jugador pueda volver a interactuar de nuevo con el juego.

4. Las acciones son **inmediatas**. Esto implica que dos unidades nunca pueden recibir órdenes al mismo tiempo. A efectos de juego, primero deberá realizarse una acción con una de las dos, esperar a que dicha acción se resuelva, y luego realizar la acción con la segunda.

5. Las reglas son **idénticas** para el jugador y la máquina. A efectos de juego, los dos pueden realizar las mismas acciones y estas se resuelven de manera similar independientemente de quién las use.

6. El juego **no requiere habilidad** para ganar. La dificultad del juego se basa en un ejercicio mental para diseñar la mejor táctica, por lo que la rapidez o soltura del jugador no influyen a la hora de evaluar su ejecución.

8.2. Visión general

Se ha decidido que el nombre del videojuego sea "Teria Wars". El juego cuenta con varios niveles que van aumentando progresivamente de dificultad, ofreciendo cada vez retos estratégicos más complicados para el jugador.

8.2.1. Aspecto general del juego

Figura 8.1: Aspecto general del juego

La imagen 8.1 muestra el ejemplo de una batalla estándar. Se puede observar como el "tablero"de juego está compuesto por una serie de casillas, cada una mostrando un tipo de terreno distinto. Aunque las casillas no están delimitadas por líneas, se puede intuir fácilmente cuanto ocupa cada una. Sobre algunas de las casillas aparecen distintas unidades. Las unidades verdes pertenecen al jugador, mientras que las unidades rojas pertenecen al adversario.

En el ejemplo, una de las unidades ha sido seleccionada, lo cual se puede advertir al observar el color de dicho soldado, que está ligeramente iluminado. Alrededor de la unidad seleccionada se han coloreado algunas casillas, que indican el rango de acción de dicha unidad. También se puede observar que se ha pinchado sobre una casilla en concreto (la que tiene un número escrito), desplegando un menú lateral con las distintas órdenes que se le pueden dar a la unidad seleccionada.

8.2.2. Ambientación

El juego está ambientado en un mundo ficticio en el que dos naciones opuestas se encargan de librar una guerra constante. El jugador toma los mandos de una de las dos naciones, Teria, liderando a las unidades para ganar las distintas batallas que se le van presentando. Al principio de cada batalla se ofrece al jugador una breve sucesión de mensajes cuyo único objetivo es la inmersión en dicha ambientación, y en el fondo narrativo de la batalla que se va a jugar.

8.3. Reglas generales

El videojuego sigue una serie de reglas y procesos que es importante conocer antes de empezar a explicar la IA:

8.3.1. Objetivo del juego

El objetivo del juego es sencillo: Acabar con el adversario. Tanto el jugador como la máquina moverán sus unidades y atacarán con ellas hasta que el ejército enemigo haya sido completamente derrotado.

Condiciones de victoria

Si al final de un turno cualquiera no quedan unidades enemigas, el juego mostrará un mensaje de victoria y la partida acabará. Normalmente se dará paso al siguiente nivel, mostrando al jugador una contraseña para acceder a este en el futuro.

Figura 8.2: Pantalla de victoria

Condiciones de derrota

Si al final de un turno cualquiera no quedan unidades aliadas, el juego mostrará un mensaje de derrota y la partida acabará. Normalmente se volverá a cargar el mismo nivel, para dar otra oportunidad al jugador de obtener la victoria.

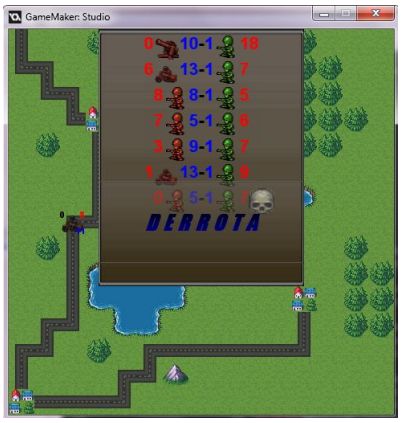


Figura 8.3: Pantalla de derrota

8.3.2. Controles

El jugador interactúa con el juego usando el ratón y pinchando en los distintos elementos del juego. En niveles grandes, se pueden usar las teclas de dirección para moverse por el mapa y la barra espaciadora para mostrar el mapa al completo. Otras teclas pueden ser utilizadas para introducir contraseñas.

8.3.3. Sistema de turnos

El juego se basa en un sistema de turnos, alternando entre el jugador (al que siempre corresponde el primer turno) y el enemigo, controlado por la IA.

Turno de jugador

En el turno del jugador, este tiene un tiempo limitado para seleccionar sus distintas unidades y darles órdenes. Cada orden se resuelve inmediatamente. El jugador puede pasar el turno cuando lo desee, dándolo por finalizado.

Turno de la IA

En el turno del enemigo, la IA se encarga de analizar el terreno y las unidades, y ejecuta las órdenes de cada una de estas de forma automática. El jugador puede ver cómo se ejecutan dichas órdenes, aunque los procesos de evaluación y toma de decisiones se ejecutan internamente. Una vez todas las unidades del enemigo han terminado de resolver sus acciones, se devuelve el control al jugador. La única acción que puede ejercer este último durante el turno de la IA es la de mover la vista, si el mapa es demasiado grande, pero esta se centrará en cada unidad cuando se ejecute una acción con ella.

8.3.4. Características de unidades

Las unidades tienen distintos atributos, recogidos en variables, que marcan las diferencias entre ellas y establecen el estado de dicha unidad durante el juego. Hay algunas características inmutables, mientras que otras dependen del estado de dicha unidad en un momento concreto.

Tipo

El tipo de unidad es un valor numérico que la identifica como un tipo de unidad u otro. Por ejemplo, dos unidades de tipo "Soldado"tendrían el mismo valor numérico en este atributo, pero este sería diferente al de una unidad de tipo "Tanque". Este valor es interno y no cambia durante el juego.

Bando

El bando de la unidad es un valor numérico que identifica a esa unidad como aliada o enemiga. Una unidad puede ser parte del ejército del jugador, en cuyo caso será este el que la controle, o del ejército de la máquina, en cuyo caso será la IA la que decida sus órdenes. Es un valor inmutable.

Vida

La vida de la unidad específica cuantos puntos de aguante le restan a esa unidad, siendo el máximo 100 y el mínimo 1. Una unidad cuyos puntos de vida descienden por debajo de 1 es eliminada automáticamente del juego. La vida también influye en la potencia de la unidad a la hora de atacar: Cuanta más vida le reste a esa unidad, más daño hará en combate.

Acciones

El valor numérico que indica las acciones de la unidad se recarga cada turno hasta un máximo indicado en los parámetros de dicha unidad, si estaba por debajo. Al ejecutar órdenes, la unidad consume un número determinado de acciones que se restan del total restante para ese turno. Cada orden conlleva un coste de acciones determinado que depende de la unidad y, si se mueve, del terreno por el que pasa. Una unidad que no tiene suficientes acciones para realizar una orden no puede usar dicha orden.

Coste de ataque

Este valor indica el coste de acciones que le supone a la unidad realizar un ataque.

Rango de ataque

Este valor indica la distancia a la que la unidad puede atacar. Esta distancia se mide en casillas, y es el número de casillas que hay en el camino más corto entre las dos unidades, moviéndose de forma ortogonal y teniendo en cuenta la casilla de la unidad objetivo (pero no la de la unidad atacante). Este rango es normalmente uno, indicando que la unidad puede atacar a cualquier otra que se encuentre adyacente (de nuevo, ignorando las diagonales). Sin embargo, en ocasiones puede ser superior, e incluso tener un rango mínimo, en cuyo caso la unidad no podría atacar si su objetivo se encuentra a una distancia inferior a dicho mínimo.

Efectividad de ataque

Cuando una unidad ataca a otra, el valor numérico que resta a la vida de su objetivo depende de una tabla en la que se indica cuál es la efectividad de dicho ataque. La efectividad depende del tipo de unidad atacante y del tipo de unidad defensora, así como de la vida restante de la unidad atacante y la cobertura de la defensora. Una unidad puede tener distintos tipos de efectividad contra tipos distintos de unidades. Por ejemplo, un soldado podría ser bastante efectivo contra otro soldado

(reduciendo en gran cantidad su vida), mientras que contra un tanque su efectividad podría verse mermada (quitando mucha menos vida a este último).

Descripción

Todas las unidades cuentan con una descripción que ofrece detalles sobre su uso habitual, sus puntos fuertes y sus debilidades.

8.3.5. Características de terreno

Los distintos tipos de terreno contienen ciertos atributos que influyen en las unidades que se encuentran en dicho terreno o que lo cruzan con su movimiento.

Coste de movimiento

Cada unidad tiene cierto coste de movimiento asociado al pasar por un terreno. Este coste es el número de acciones que se le restan cuando se mueve a dicha casilla. Si el coste de movimiento es mayor al número de acciones restantes de dicha unidad, entonces la unidad no puede moverse hacia ese punto.

Cobertura

Cada tipo de terreno tiene un valor de cobertura asociado. Este valor reduce proporcionalmente el daño recibido al sufrir un ataque en dicho terreno. Esto es, si un soldado que se encuentra en un terreno de tipo ciudad recibe un ataque por parte de otra unidad, el valor del ataque sería reducido proporcionalmente teniendo en cuenta la cobertura que ofrecen los terrenos de tipo ciudad.

Descripción

Cada terreno tiene su descripción, en la que se explica cuáles son las ventajas y desventajas tácticas de dicho terreno.

8.3.6. Acciones permitidas

Hay varias acciones que tanto la IA como el jugador tienen permitidas durante su turno. Se pueden dar todas las órdenes que se desee a las unidades, siempre y cuando estas tengan suficientes acciones para llevarlas a cabo.

Mostrar información

La opción de mostrar información siempre está presente al desplegar el menú, y no consume acciones en caso de que hubiera una unidad elegida. Esta acción genera un cuadro de diálogo en el que se muestra la descripción de o bien la unidad o el terreno de la casilla sobre la que se ha desplegado el menú previamente o, si el tamaño de la ventana lo permite, de los dos.

Figura 8.4: Ejemplo de mostrar información

Mover

Esta acción se muestra al elegir una unidad y pinchar en alguna zona a la que dicha unidad pueda acceder con su movimiento. El movimiento tiene un coste igual a la suma de los costes de las casillas del camino recorrido, para esa unidad. El camino recorrido para llegar a una casilla que no se encuentra adyacente a la unidad (sin contar las diagonales) es siempre el que supone menor gasto de acciones. Las unidades pueden moverse a cualquier casilla a la que puedan llegar con sus acciones, pero no podrán acceder a casillas en las que el camino más corto tenga un coste de acciones superior a las acciones restantes de la unidad.

Las unidades pueden atravesar casillas que contengan otras unidades aliadas, pero nunca aquellas que contengan una unidad enemiga. Una unidad no puede acabar su movimiento en una casilla ocupada por otra unidad. Esto implica que bajo ningún concepto puede haber dos unidades que ocupen la misma casilla de terreno al mismo tiempo.

Como ayuda visual, al seleccionar a una unidad se iluminarán en verde todas las casillas a las que puede acceder dicha unidad con sus acciones restantes. Al pasar el puntero por cualquiera de esas casillas se mostrará un valor numérico en el centro de la casilla. Este valor numérico es otra ayuda visual que indica las acciones que le quedarían a la unidad seleccionada en caso de mover a esa casilla. Si el valor numérico está resaltado en rojo indica que dicha unidad aún tendría acciones suficientes para realizar al menos un ataque después de haber movido a dicha casilla.

Figura 8.5: Ejemplo de movimiento

Atacar

La orden de ataque incluye a la unidad seleccionada y a la unidad elegida posteriormente. La unidad atacada perderá una cantidad de vida aleatoria cuyo valor varía entre dos límites (inferior y superior) dados por la efectividad de ataque de la unidad atacante contra ese tipo de unidad. La cantidad de vida perdida también se ve modificada por la vida restante de la unidad atacante y la cobertura de la unidad defensora, como ya se ha explicado previamente.

Siempre que la unidad atacada continúe con vida tras recibir el impacto, y si la unidad atacante se encuentra dentro de su rango de ataque, puede darse lugar a un contraataque. Este efecto es automático, e implica que la unidad atacada podrá atacar a su vez a su agresora, aunque con la efectividad reducida (y siempre después de aplicar la reducción sobre su vida). Exceptuando el detalle de que la efectividad es siempre menor que la de un ataque normal, y de que un contraataque nunca genera otro contraataque, el resto de reglas que se aplican son las mismas que las de un ataque normal.

Una unidad solo puede atacar a otras que pertenezcan a un bando distinto al suyo. Tampoco puede atacar a otra unidad si esta no se encuentra dentro de su rango de ataque. Para facilitar una ayuda visual, al seleccionar una unidad aliada se marcarán en rojo las casillas a las que dicha unidad puede atacar; es decir, todas aquellas que contengan una unidad de otro bando y se encuentren dentro del rango de ataque.

También se marcarán en rojo todas aquellas unidades que, a pesar de no estar dentro del rango de ataque, puedan llegar a estarlo si la unidad atacante se mueve antes a una casilla específica, y siempre que después de ese hipotético movimiento la unidad atacante aún tuviese suficientes acciones para realizar un ataque. En este último caso, se marcará la casilla a la que cueste menos acciones moverse para poder atacar a dicha unidad, con un punto de mira.

Al ordenar un ataque, la unidad realizará el ataque de forma inmediata o, si necesitaba moverse antes, primero moverá hasta la casilla marcada con el punto de mira y luego atacará. Esto no siempre es la mejor opción, puesto que la posición se elige solo por el coste de acciones, y no por las ventajas tácticas de la casilla de destino, como la cobertura o la posición con respecto a otras unidades. Por tanto, si el jugador quiere atacar desde otra casilla distinta, primero deberá mover manualmente a la unidad hasta dicha casilla, y luego darle una nueva orden, esta vez de ataque.

Figura 8.6: Ejemplo de ataque

Guardia

Una unidad que hace guardia invierte todas sus acciones restantes en reforzar su posición, mejorando la cobertura de su terreno temporalmente en un valor que depende del porcentaje de acciones que le quedara respecto a su total. Por tanto, una unidad que todavía no hubiera gastado ninguna acción obtendrá mayor beneficio al hacer guardia que otra que ya hubiera consumido un cuarto de sus acciones. Al ser atacada en el turno del adversario, una unidad que hubiera hecho guardia contará con el modificador añadido a la cobertura que hubiera logrado en su turno. Sin embargo, este modificador desaparecerá por completo al empezar el nuevo turno del bando de dicha unidad.

Pasar turno

La acción de pasar turno puede usarse aunque no se haya seleccionado una unidad. Esta acción finaliza el turno del jugador y se lo cede al bando enemigo. Todas las unidades aliadas que todavía tuvieran acciones restantes por utilizar realizan inmediatamente y de forma automática una orden de guardia, dejando por tanto sus acciones a cero antes de pasar el turno y consiguiendo el modificador a su cobertura.

8.3.7. Otras reglas

Hay otras reglas y peculiaridades del juego que es importante señalar.

Figura 8.7: Tutorial del primer nivel del juego

Mensajes

Al principio de cada nivel se sucederán una serie de cuadros de diálogo que ofrecen descripciones sobre la batalla. Estos mensajes suelen tener un mero valor estético, ofreciendo detalles sobre el trasfondo y la ambientación, aunque en ocasiones también proporcionan información relacionada con las mecánicas del juego o con los detalles específicos de dicho nivel. Estos cuadros de diálogo pueden saltarse pulsando la tecla 's'.

Menú principal

El menú principal es precedido por una breve animación de texto, tras la cual se muestra una lista con los niveles a los que se puede acceder. Inicialmente solo se puede acceder al primer nivel, pero según se vayan superando las distintas batallas o se introduzcan contraseñas se dará acceso a nuevos niveles.

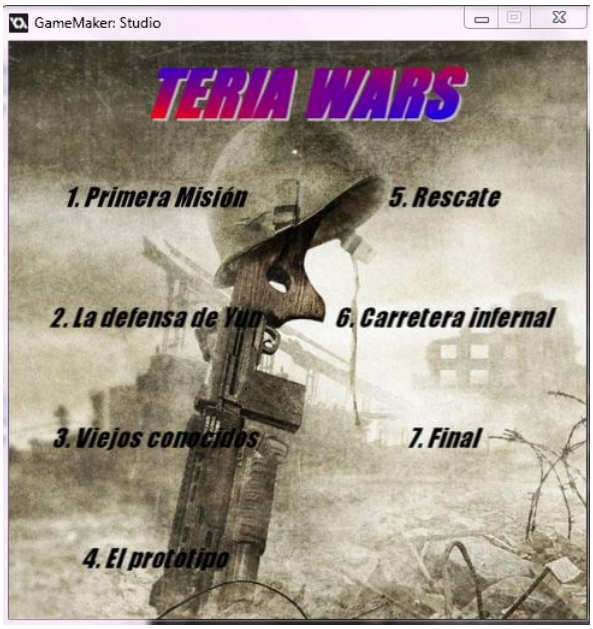


Figura 8.8: Pantalla de selección de nivel

Contraseñas

Al final de cada batalla se ofrece una contraseña, si el jugador ha conseguido la victoria, que da acceso al siguiente nivel. Dicha contraseña se puede introducir pulsando la tecla 'Shift' y luego una sucesión concreta de letras que formen la palabra específica de dicha contraseña.

8.4. Tipos de unidades

Los tipos de unidades se describirán por encima. Existen en total quince clases distintas de unidad.

8.4.1. Unidades de infantería

Las unidades de infantería son la base de todo ejército. Aunque relativamente débiles contra otro tipo de unidades más blindadas o poderosas, son versátiles y su movilidad les permite realizar todo tipo de funciones, desde la protección hasta el control de casillas clave.

Soldado

La unidad más básica de cualquier ejército, es vulnerable frente a otros tipos de unidades, pero efectiva contra otras unidades de infantería.

Mecanizada

Una unidad especializada contra vehículos y tanques pequeños, su movilidad se ve reducida frente a la de los soldados a costa de una mayor potencia de fuego.

Lanzallamas

Esta unidad está especialmente equipada para enfrentarse a otras unidades de infantería, mejorando su efectividad contra estas en gran medida a cambio de perder movilidad y efectividad contra otros tipos de unidades.

8.4.2. Vehículos

Los vehículos son rápidos y efectivos contra otras unidades poco blindadas, pero su movilidad se ve reducida en terrenos difíciles.

Jeep

Los jeep son efectivos contra unidades de infantería y otros vehículos, pero resultan débiles contra tanques. Son posiblemente las unidades con mayor movilidad en carretera, pero esta se ve reducida en tipos de terreno más accidentado.

Vehículo médico

Los vehículos médicos son las unidades más débiles de todo el juego. Su potencia de fuego es mínima, y son vulnerables contra cualquier otro tipo de unidad.

8.4.3. Tanques

Si la infantería es la base de cualquier ejército, los tanques son su corazón. Suficientemente blindados para resistir los ataques más fuertes y con menores problemas que otros vehículos para atravesar terrenos accidentados, su potencia de fuego suele ser demoledora.

Tanque pequeño

La versión más simple del tanque, su movilidad es alta incluso en terrenos difíciles. Es efectiva contra todo tipo de unidades.

Tanque mediano

La evolución del tanque pequeño, su movilidad es mayor pero su potencia y blindaje frente a los ataques aumentan en gran medida.

Tanque blindado

Una modificación del tanque mediano que añade mucho más blindaje y mejora ligeramente la potencia a cambio de sacrificar algo más de movilidad, convirtiendo a esta unidad en un enemigo extremadamente difícil de abatir.

8.4.4. Unidades de artillería

Las unidades de artillería se caracterizan por su alcance. A diferencia de otras unidades, su rango de ataque les permite atacar a unidades que no son adyacentes. Por desgracia, su movilidad es baja.

Artillería

Unidad básica de artillería, su efectividad contra todo tipo de unidades es alta, pero también es vulnerable al ataque de otras unidades.

Gran cañón

El gran cañón es una unidad con una potencia muy alta, que resulta extremadamente efectiva contra unidades de todo tipo, y cuenta con mayores defensas que la artillería. Sin embargo, su alta complejidad le impide moverse y atacar en el mismo turno.

8.4.5. Mixtas

Estas unidades tienen características comunes con dos grupos de unidades, lo que las sitúa en un punto intermedio entre dichos grupos.

Francotiradores

Aunque son esencialmente infantería, pueden atacar a distancia con gran efectividad, por lo que en muchas ocasiones cumplen una función similar a las unidades de artillería. Son vulnerables a los ataques enemigos.

Fuerzas especiales

Esta unidad de infantería es efectiva contra todo tipo de unidades, pero vulnerable al ser atacada. Al atacar ella, el contraataque que recibe siempre tiene una efectividad mínima. Sin embargo, cuando es atacada, la efectividad de sus contraataques también se reduce significativamente. Tienen un rango de ataque ampliado, lo que les permite atacar a unidades adyacentes o a dos casillas de distancia.

Motorizadas

A medio camino entre la infantería y los vehículos, las unidades motorizadas compensan su baja efectividad contra otros vehículos o tanques con su gran movilidad, en especial por carretera. A efectos de potencia son comparables a los soldados, pero ligeramente mejores.

Orugas

Las orugas son, a todos los efectos, un vehículo con el blindaje y la movilidad de un tanque. Su efectividad contra otras unidades es similar a la de los jeeps.

Tanque X

La unidad más poderosa del juego, y también la más lenta. Es más efectiva que un gran cañón y más resistente que un tanque blindado. Su movimiento es extremadamente reducido, pero puede atacar hasta a dos casillas de distancia.

8.5. Tipos de terreno

La importancia estratégica de un tipo de terreno radica en el coste de movimiento que deben invertir las unidades para cruzarlo, y en la cobertura que ofrecen.

8.5.1. Carretera

La carretera no ofrece ningún tipo de protección, pero el coste de movimiento será siempre el mínimo para todas las unidades.

8.5.2. Pueblo

Los pueblos son núcleos urbanos pequeños, que ofrecen una protección considerable y además no obstaculizan el movimiento de las unidades.

8.5.3. Ciudad

Las ciudades son puntos estratégicos muy valiosos, pues no solo tienen el coste mínimo de movimiento para todas las unidades, sino que ofrecen una de las mayores coberturas del juego.

8.5.4. Llano

Los llanos son terrenos fáciles de cruzar, que obstaculizan levemente el movimiento de algunos vehículos y tanques pero ofrecen una mínima cobertura frente a ataques enemigos.

8.5.5. Bosque

Los bosques ofrecen una buena cobertura frente a ataques enemigos, pero obstaculizan el movimiento de muchas unidades.

8.5.6. Montaña

Las montañas ofrecen una de las mayores coberturas del juego, pero también es uno de los terrenos que más obstaculiza a las unidades a la hora de cruzarlo, llegando a ser incluso inaccesible para algunas unidades especialmente pesadas o aparatosas.

8.5.7. Desértico

El terreno desértico es el más vulnerable y peligroso, pues no solo ofrece una cobertura nula, sino que penaliza el movimiento de todas las unidades.

8.5.8. Agua

El agua es un tipo de terreno inaccesible.

Capítulo 9

Problemas con la IA

9.1. Un entorno complejo

Al empezar a pensar en la necesidad de aplicar una inteligencia artificial eficiente para nuestro juego nos encontramos con varios problemas que, si bien por sí mismos no parecen suponer un gran obstáculo para una ejecución efectiva, en conjunto muestran varias complicaciones que se apilan para resaltar la verdadera complejidad subyacente.

Es importante pues dividir los distintos obstáculos que se vayan encontrando en el diseño de la IA en distintas partes, de tal manera que se puedan abordar dichos problemas de forma separada, pero siempre sin perder la visión global, pues las soluciones que se apliquen pueden fácilmente implicar nuevas complicaciones que al final resulten fatales y originen más perjuicios de los que han resuelto.

Como resulta habitual en estos casos, se encuentran dos tipos de metas a alcanzar bien diferenciadas, que diferenciaremos como "problemas de alto nivelz "problemas de bajo nivel". Esta calificación arbitraria sirve para clasificar y poner en perspectiva dos clases distintas de quebraderos de cabeza: Aquellas que afectan al programa general, y plantean un problema más abstracto y teórico, y aquellas que actúan a un nivel más reducido, y que a pesar de entrañar en muchas ocasiones tanta o más dificultad que los problemas de alto nivel, suelen referirse a aspectos concretos del sistema que pueden reducirse a un contexto aislado.

9.2. Problemas de alto nivel

Los problemas de alto nivel son, como se ha explicado previamente, todos aquellos que tratan el concepto de la Inteligencia Artificial como un conjunto, usando abstracciones y centrándose en los aspectos de esta a grandes rasgos, sin entrar en detalles sobre la ejecución o la implementación de dichas soluciones. Esta clasificación (que conviene aclarar de nuevo, es arbitraria) sirve para entender .el cuadro completo", y de esta manera extraer posteriormente los problemas de bajo nivel que se necesitarán superar.

9.2.1. Dirigir un ejército

Como alcanzar la victoria

Lo primero que viene a la cabeza al abordar las necesidades de la IA que se está intentando diseñar es la manera óptima de implementar dicha IA o, dicho de otra manera, la serie de directrices

y reglas que debería seguir dicha IA para lograr su cometido de la forma más satisfactoria posible. En este caso, el objetivo de la IA es dirigir un ejército de unidades en un terreno mutable, con el objetivo de acabar con el enemigo.

Esta meta entraña más complejidades de lo que podría parecer en un primer momento, pues la definición de una buena estrategia varía según la persona a la que se pregunte. Aunque está claro que los resultados son un claro indicador de la efectividad de dicha estrategia, en muchas ocasiones la suerte, representada por las decisiones que toma el rival y el propio azar que se ha decidido introducir en el combate, juega un papel decisivo a la hora de definir estos resultados. Dos estrategas podrían actuar de maneras radicalmente opuestas y conseguir resultados similares.

Esto no implica que no se pueda intentar alcanzar una definición que, en este caso, sirva para abordar el trabajo que tenemos entre manos. La reflexión anterior puede servir para señalar que, efectivamente, dicha solución será profundamente subjetiva y sujeta a debate, pero no por ello se puede excusar la necesidad de dar con una para afianzar las bases de la IA.

Una buena estrategia

No se arriesga mucho al afirmar que, dadas las condiciones de victoria presentadas previamente para el juego, un resultado deseable es la eliminación de todas las unidades enemigas. Asimismo, también es fácil deducir que la conservación de las propias unidades es un factor de éxito, sobre todo en lo referente a la consecución de objetivos parciales. Dicho de otro modo, es mejor acabar con diez unidades enemigas sin haber perdido ninguna unidad aliada a lograr lo mismo habiendo sufrido bajas entre ellas. Este podría considerarse un objetivo a corto plazo ya que, sin duda, contar con más unidades para luchar en otras zonas del terreno de juego es siempre un beneficio que probablemente permita alcanzar la victoria con mayor facilidad.

¿Cómo alcanzar estas condiciones deseadas? El medio, como era de prever, resulta algo más delicado y complejo de definir. Al fin y al cabo, atacar con fiereza en algunas situaciones podría dar lugar a caer en trampas del enemigo o perder unidades importantes por beneficios mediocres. Por otro lado, en otras ocasiones actuar con excesiva prudencia puede dar lugar a que el enemigo tenga tiempo para colocar a sus unidades en posiciones ventajosas, lo que a largo plazo podría suponer una derrota segura.

No parece que haya parámetros absolutos a la hora de definir una buena estrategia, e intentar representar los refinados mecanismos que se dan en la mente humana a la hora de tomar decisiones tan complejas como las mencionadas antes queda un poco por encima del alcance de este proyecto, y deberá dejarse a un lado para otros más ambiciosos.

Sin embargo, si es posible simular algunos de estos comportamientos, a nivel básico, e intentar aplicarlos cuando se den ciertas condiciones. El sistema así propuesto distaría mucho de ser tan efectivo como la mente de una persona, pero podría llegar a ofrecer retos y a demostrar comportamientos emergentes interesantes.

Sin embargo, permitiéndose la indulgencia de simplificar el conjunto de forma algo burda aunque necesaria, se podría decir el corazón de la estrategia se basa en el conjunto de las acciones de cada una de las unidades que componen el ejército. Dicho conjunto es, al final, el que genera los resultados finales, y aunque es importante tener siempre en mente que, como conjunto, no son entidades independientes, sí que resulta recomendable centrarse en cada una de estas acciones a un nivel más individual, para luego llevar las conclusiones a la visión colectiva.

9.2.2. Acciones

En el juego, cada unidad podría definirse como un ente independiente que, a pesar de estar estrechamente relacionado con otras unidades, actúa y ejerce cambios en el estado de juego de forma individual y directa. Cada turno, dicha unidad tiene a su disposición una serie de opciones entre las que puede elegir las que más le convengan (o las que más convengan a su bando) para lograr objetivos. Los objetivos individuales de cada unidad podrían ser tan sencillos como "destruir a tal unidad enemiga.º .ªscoltar a una unidad aliada". Sin embargo, estos objetivos individuales tienen repercusión en aquellos objetivos generales del bando, normalmente favoreciéndolos de forma directa.

Deseabilidad de las acciones

Elegir el curso de ejecución más adecuado para una unidad puede ser relativamente fácil, midiendo una serie de factores que asignen cierta prioridad a cada movimiento o acción posible y controlando el entorno más cercano a la unidad. Tras analizar el terreno cercano y situar las posiciones de aquellas unidades amigas o enemigas que se encuentren en él, se podría crear una lista con las distintas opciones disponibles de cara a dichas unidades y a dicho terreno, asignándole a cada una un valor arbitrario dependiente de lo deseable del resultado.

Esta situación se visualiza mejor si se concreta. Por tanto, resulta interesante llevarla a la práctica, enunciando una hipótesis. Así, por ejemplo, una unidad rodeada de otras tres, dos de ellas en su mismo bando y otra enemiga, podría decidir tres acciones distintas, a saber:

1. Atacar a la unidad enemiga.

2. Seguir a una unidad aliada.

3. Cubrir a la otra unidad aliada bloqueando el camino de la unidad enemiga hacia ella.

Estos tres ejemplos podrían ser considerados acciones estratégicas válidas, cada una orientada a conseguir cierta ventaja táctica sobre el enemigo:

1. La reducción de los efectivos del enemigo en la primera, ya que al mermar las fuerzas de dicha unidad enemiga, el ejército rival pierde recursos que podría utilizar más adelante para mermar las propias fuerzas aliadas.

2. En la segunda, la búsqueda de una efectividad mayor por medio de la unión de distintos agentes. Al permanecer en posiciones cercanas, pueden auxiliarse mutuamente y protegerse de ataques enemigos posteriores, o atacar en conjunto para aprovechar su ventaja numérica frente a otras unidades solitarias.

3. La tercera opción implica reducir los daños de un hipotético ataque contra la unidad aliada, sufriendo dichos efectos la unidad que protege en vez de su protegida. Esto puede ser especialmente útil en los casos en los que se pueda permitir el sacrificio de una unidad débil o poco útil a cambio de conservar otra con mayor potencia o versatilidad.

Obviamente, para cada una de ellas habría que calcular parámetros adicionales, como la mejor ruta para atacar al enemigo o la posibilidad de bloquear el camino de este. Sin embargo, esos problemas corresponden al ámbito de bajo nivel que se ha definido al principio de este capítulo, y por tanto serán abordados más adelante.

Elegir acciones

Sin embargo, antes de calcular los parámetros mencionados arriba hay que decidir qué opción se va a realizar. Para ello, como se ha explicado antes, asignaríamos un valor arbitrario a cada alternativa. Este valor podría depender de varios factores, como la importancia de la unidad enemiga y de las unidades aliadas, la efectividad de ataque de la unidad que está siendo elegida, la distancia a la que se encuentra cada uno de los puntos de interés...

Como se puede deducir por esto, el movimiento táctico de una unidad encierra ciertos interrogantes que nos veremos obligados a resolver. Hay que tener en cuenta, además, que no nos basamos en un soporte de capacidad ilimitada: Nuestras limitaciones de tiempo y memoria son constricciones que debemos tener en cuenta si queremos ofrecer un sistema de toma de decisiones adecuado a nuestros requisitos.

9.2.3. Recursos limitados

Los límites con los que nos encontramos son variados y están siempre presentes, pero puede ser de utilidad intentar definir los más obvios o recalcables para tenerlos en mente cuando se aborden otros problemas.

Percepción de profundidad

Para empezar a listar los límites con los que nos encontramos, se puede empezar por aquellos que resultan intrínsecos para cualquier sistema de estas características. El primero y más obvio, es que nuestra máquina carece de visión de profundidad, lo que nos impide juzgar los alrededores de la unidad de una manera fiable si no dedicamos tiempo y recursos al mapeo y consecuente identificación de puntos críticos. Aunque este problema se puede abordar de muchas maneras, es uno de los principales aspectos que diferencia la mente humana del procesador de cualquier ordenador.

Por utilizar una analogía algo poética (y que, por tanto, debe de ser analizada con cierta holgura), una persona podría mirar por una ventana y hacerse una idea inmediata de las consecuencias de caer al suelo desde esa altura. En la misma situación, una máquina debería analizar la altura recorriendo cada piso uno por uno. Esta falta de análisis inmediato hace que decisiones que para un humano resultarían sencillas (como, por ejemplo, encontrar el camino más rápido para bordear una roca), para una máquina se conviertan en verdaderos retos intelectuales.

Falta de criterios

Una vez tenemos dicha información, todavía nos queda compararla según algunos criterios fijos para decidir qué tipo de acciones tenemos a nuestra disposición. Dichos criterios no son innatos para la IA, y han de ser definidos de forma exhaustiva para que esta no cometa errores fatales. Aunque para muchas personas sería obvio que un soldado no debe cargar contra un tanque pesado que está rodeado de unidades de artillería, para nuestra IA dicha deducción se reduce a una serie de comparaciones y números que conllevan una salida fija (pues tampoco es capaz de improvisar nuevas soluciones).

Una vez hemos identificado dichas acciones y criterios para decidir la utilidad de los resultados, todavía queda volverlas a comparar entre sí según el resultado esperado de cada una y el valor heurístico que hemos asociado a estas para decidir, por último, qué es lo más deseable. Esta comparación tampoco es sencilla, pues se deben descartar en muchas ocasiones alternativas que en otros casos podrían resultar útiles, pero en el estado en el que se encuentra el juego en ese instante podrían resultar fatales.

Parece suficientemente complejo de por sí, pero se cuenta con la ventaja de poder decidir qué clase de criterios se van a utilizar, pudiendo simplificar el proceso de manera que la simulación de una mente estratégica resida en realidad sobre una base de comportamientos genéricos fijados, aplicables a una gran variedad de situaciones.

Quizás este concepto se aleje un poco de la filosofía inicial, que supone huir de las recetas para todoz encontrar un tipo de procesamiento dinámico que permita evaluar y actuar en cada situación sin plantillas de ningún tipo. Por otro lado, ahorra gran cantidad de tiempo y recursos para evitar que el entorno se convierta en algo imposible, con un coste demasiado alto para tener utilidad real de cara al jugador.

Limitaciones de tiempo

Otro factor que es esencial recordar es el del tiempo. En el juego, se espera que el oponente máquina pueda tomar decisiones en un rango aproximado de tiempo, ya que el tardar demasiado podría entorpecer de forma significativa la experiencia de juego, pudiendo llegar a hacerlo injugable. Este es probablemente uno de los mayores "murosçon los que se puede llegar a encontrar la IA, pues reducir el tiempo con el que se cuenta para tomar decisiones implica también reducir la cantidad de operaciones que se pueden realizar y, por tanto, también la optimización de dichas decisiones.

Por supuesto, siempre se pueden utilizar técnicas o pequeños "trucos"para conseguir algo más de tiempo, como mostrar al usuario lo que ocurre en todo momento en tiempo real. Esto le permite observar lo que está ocurriendo y comprenderlo en vez de tener que deducirlo del resultado final, lo que permite disponer de una mayor cantidad de tiempo sin hacer sentir al usuario incómodo o aburrido.

Limitaciones de recursos

La memoria de la que se dispone para guardar datos que utilizar luego en la toma de decisiones también en finita, y aunque este aspecto suele originar considerablemente menos problemas que el del tiempo, es también algo que debe tenerse en cuenta, pues un claro abuso de este tipo de recursos podría llegar a saturar el programa, haciendo que vaya más lento o funcione de manera imprevista.

9.2.4. Cooperación

Una vez tratados los aspectos más básicos, como son las acciones de cada unidad y los requisitos del soporte en el que se va a realizar el proyecto, queda plantearse si los interrogantes que puedan surgir de aquí en adelante no serán más sencillos de abarcar.

Un espacio compartido

Sin embargo, hasta ahora se ha obviado mencionar uno de los principales escollos con los que se debe enfrentar la IA: ¿Qué ocurre cuando nos encontramos con varias unidades de nuestro bando a las que tenemos que dar órdenes? Un sistema de toma de decisiones que se encargara únicamente de una sola unidad, la cual actuara como un agente independiente, probablemente requeriría una cantidad de recursos y esfuerzo menor del que se necesita para organizar a varios agentes que cooperan entre sí para conseguir un fin común.

En el supuesto anterior, las dos unidades aliadas también forman parte de nuestro bando. ¿Por qué, entonces, escogemos primero a la unidad que habíamos elegido para realizar sus acciones? Un sistema de IA eficiente no dejaría esta decisión al azar, y aunque para cualquier persona la decisión

podría ser trivial, se observa que de cara a la computación esta elección resulta más compleja de lo previsto.

Orden de acciones

Es por tanto necesario centrarse en el orden de las acciones, y en cómo se va a abordar el dilema. Para empezar... ¿Qué criterio se usa en tal decisión? Se podría argumentar que la cercanía a las unidades enemigas resulta un factor decisivo, pero pronto observamos distintos fallos en este razonamiento. Uno es la dispersión del ejército enemigo: Normalmente las unidades del bando contrario no formarán en una figura compacta, sino que se mantendrán separadas las unas de las otras, cada una moviéndose de forma independiente. Esto puede implicar que las unidades del bando aliado aparezcan intercaladas en el mapa con las del bando enemigo. ¿Cuál está más cerca del grueso del ejército, si este se encuentra separado de forma heterogénea?

Otro criterio sería elegir a las unidades más ïmportantes"primero, usando una lista de prioridad mediante la cual asignemos un valor de preferencia alto a aquellos tipos de unidad más útiles o más poderosos, mientras que aquellas más dispensables se sitúen en el final de dicha lista. Por tanto, empezaríamos dando órdenes a nuestras "fichas"más valiosas. En el ajedrez, sería el equivalente a empezar moviendo el rey o la reina, continuando con los alfiles, torres y caballos, y terminando con los peones. Este sistema puede parecer algo más útil, pero sigue sin parecer del todo óptimo.

Se plantean más criterios aún, pudiendo ser usados de forma única o para eliminar la ambigüedad que pueda crearse al utilizar otros. Empezar a elegir unidades de un lado del mapa al otro, hacerlo al azar, tener en cuenta parámetros variables como la vida, las acciones o la cobertura, elegir unidades que hayan entrado en conflicto en turnos anteriores, etcétera...

Sin embargo, todos y cada uno de estos criterios se limitan a resolver la cuestión de qué unidad elegir primero, sin mencionar otros problemas que podrían resultar aún más decisivos.

9.2.5. Solapamiento de órdenes

Acciones que entorpecen

Si se vuelve al ejemplo anterior, se puede comprobar con facilidad cuál será el gran obstáculo con el que la IA se enfrentará. Al ponerse en el supuesto de que se elige una de las tres unidades disponibles para ejecutar una acción, se podría argumentar que, cumpliéndose ciertas condiciones, el mejor curso de acción para dicha unidad de forma individual sería atacar a la unidad enemiga. Para ello, de nuevo, se podría aventurar que la mejor idea es elegir el camino más corto, ya que el gasto de acciones es menor y eso nos permite luego realizar otras posibles acciones, como escapar del conflicto y moverse a un terreno con mayor cobertura después de atacar.

Siguiendo dicho hilo de pensamiento, el foco de atención pasa a otra de las unidades aliadas, que también decide que el mejor curso de acción es atacar a la unidad enemiga. Sin embargo, al comprobar la posibilidad de esta acción, se descubre que la única posición que podría alcanzar con sus acciones restantes para atacar a la unidad enemiga es la que ahora ocupa la primera unidad que se utilizó en este turno. En retrospectiva, la primera unidad podría haber elegido un camino algo más largo, que también le habría permitido atacar, dejando así espacio para que la segunda pudiera atacar a su vez.

Este es un claro ejemplo de las desventajas que la IA podría tener si solo se centrara en las unidades de forma individual. Para establecer tácticas efectivas se requiere no solo de una evaluación del entorno, sino de un proceso de cooperación entre las distintas unidades para elegir el curso de acciones que permitan, en su conjunto, un resultado global deseable. Al centrarse en una solitaria unidad se puede obtener un beneficio individual para dicha unidad, como eliminar un enemigo u

obtener una cobertura adecuada, pero al mismo tiempo se podría caer en el error de causar un perjuicio al bando, como bloquear un camino que otras unidades necesiten traspasar para completar un objetivo global o perder una cantidad de vida en el combate que podría haberse paliado al ejecutar una unidad más poderosa esa misma acción.

Evitar el solapamiento

¿Cómo conseguir que las unidades se coordinen entre sí, cooperen y muestren un táctica unificada que les permita completar objetivos en conjunto, en vez de entorpecerse y actuar en solitario? No hay una fácil respuesta, pues el entorno, como se ha explicado antes, es complejo y dinámico. Si se piensa en cada unidad como en un ente separado, es inevitable que esta solo pueda realizar un proceso lógico centrado en ella misma, sin conocer de antemano qué se proponen las otras unidades que le rodean. Viene a la mente la posibilidad de usar un sistema de señales entre unidades, pero dicho sistema sería bastante complejo y consumiría muchos recursos en caso de ser realmente eficiente.

Otra posibilidad sería tratar el tablero como un todo, de forma similar a los programas de ajedrez, intentando buscar una serie de soluciones globales moviendo todas las "fichas"según un plan general sin centrarse en la mejor solución individual para cada una. Esta solución parece óptima, pero al reflexionar sobre ella se advierte que el coste y la complejidad serían excesivamente altos en mapas grandes con múltiples unidades presentes en cada bando. Al intentar que el sistema sea lo más eficiente posible sin sacrificar jugabilidad, tal posibilidad queda descartada.

9.2.6. El coordinador

La figura del coordinador

Observamos, sin embargo, que, como en muchos otros casos, el problema puede encontrar una analogía en el comportamiento humano cotidiano. ¿Cómo se coordinan un grupo de trabajadores en una empresa? Cada uno es un ente independiente con sus propios objetivos y análisis de su entorno, y sin embargo en muchas ocasiones cooperan para establecer una serie de objetivos comunes y lograr completar ciertas tareas que por sí solos no podrían realizar.

En parte, esto se debe a la comunicación entre los distintos entes. Pero, como hemos observado antes, esta solución no nos resulta demasiado útil en este entorno, pues nuestras unidades son considerablemente más simples que los sistemas complejos que interfieren en la mente humana de cualquiera de los trabajadores de dicha supuesta empresa. Sin embargo, sí presentan una ventaja que estos últimos no tienen: Obediencia absoluta. Las unidades de nuestro sistema están programadas para obedecer todas nuestras órdenes de la manera más eficiente posible. Esta es la razón principal por la cual resulta obvio llegar a la conclusión que permitirá un funcionamiento óptimo del equipo: El coordinador.

Dicha figura, siguiendo con la analogía de la empresa, normalmente viene representado por el jefe de proyecto o el delegado del departamento, dependiendo del tipo de empresa. En nuestro problema, podríamos considerar la utilización de varios "jefes de equipo", encargados de una zona reducida del mapa, que se centren en buscar la mejor solución para las unidades a su cargo en dicha zona. De esta manera, podemos buscar un compromiso entre el enfoque individual y nada cooperativo de centrarnos en cada unidad, y el enfoque global y nada eficiente de centrarnos en todas a la vez.

Elegir al coordinador

Por desgracia, se vuelve a la misma encrucijada que antes: ¿Cómo elegir a dichos coordinadores entre las decenas de unidades con las que se pueden llegar a contar? El coordinador se encarga, como

ya se ha insinuado, de ordenar a las unidades a su cargo, principalmente. No es indispensable que sea una unidad poderosa, pues su única función es la de permitir al sistema operar con unos valores reducidos, que reduzcan el tiempo de espera hasta unos valores asequibles.

Lo que sí podría llegar a resultar crítico para la decisión del coordinador es justamente los agentes que tiene a su cargo. ¿Cómo se eligen? Es lógico pensar que elegir agentes por cercanía al coordinador es una buena alternativa, pero... ¿Qué ocurre cuando hay un montón de unidades juntas? Es posible que, al usar únicamente el factor distancia para decidir qué unidades siguen a cada coordinador, se den casos de coordinadores con muchas unidades a su cargo, mientras que otros coordinadores podrían no tener a ninguna.

Cabe pensar entonces que un límite adecuado de unidades por coordinador es algo sino necesario, al menos sí recomendable. De esta manera, se puede evitar que el programa se sature por tener que procesar el conjunto de órdenes de un grupo excesivamente grande. Sin embargo, también es cierto que cuantas menos unidades sirvan a cada coordinador, menor es también la capacidad de cooperación. Hay que encontrar un compromiso entre eficiencia y consumo de recursos, estableciendo números arbitrarios de unidades hasta dar con el adecuado.

Para añadir una dimensión más al desafío que se plantea en la figura del coordinador, está la duda de elegir a coordinadores fijos frente a cargos de coordinador dinámicos, que se van pasando entre las distintas unidades. La primera opción parece dar varias ventajas, como estabilidad, pero puede llegar a resultar desastrosa si el coordinador es eliminado y no se busca un reemplazo adecuado. Además, también puede llegar a resultar previsible, algo que desde luego no se desea en una IA competitiva. La segunda opción elimina estas carencias, pero a cambio genera más preguntas. Por ejemplo, ¿cómo se evita que un coordinador sea elegido en varias ocasiones para el mismo turno? ¿Qué criterios se usan para elegir a dichos coordinadores? ¿Cómo se evita que el ejército se disperse, al no contar con una figura de referencia?

9.2.7. Un entorno variable

Sin duda, los problemas antes planteados ofrecerían una menor dificultad de poder contar con un estado inicial conocido y unas condiciones fijas ya planteadas desde el primer momento. Esto no es del todo falso, pues se conoce el tipo de unidades y atributos de cada una a la hora de diseñar la IA. Sin embargo, también es cierto que hay varios factores esenciales que se desconocen y que, por tanto, pueden variar. La IA deberá ser versátil y poder adaptarse a estas variables.

El mapa

El terreno del mapa cambia en cada nivel, y por tanto se desconocen sus características. Un mapa previamente cargado podría facilitar a la IA ciertos puntos críticos o patrones de movimiento que el diseñador introdujese previamente en el comportamiento de dicho sistema. Sin embargo, este acercamiento al problema entra en conflicto directo con la filosofía del proyecto, que pretende buscar una solución genérica y no especializada a una situación concreta pero inútil en cualquier otra.

Es por tanto importante reflexionar sobre los elementos críticos del mapa que pudieran aparecer durante el juego, y aprender a reconocerlos sobre la marcha. Terrenos infranqueables como ríos o lagos, conjuntos de terrenos específicos que ofrecen cobertura o entorpecen el movimiento, distribuciones peculiares con valor estratégico como estrechos o puntos versátiles que permiten controlar amplias zonas de terreno con facilidad son solo algunos ejemplos de este tipo de elementos que, conocidos previamente, podrían suponer una ventaja táctica considerable frente al adversario, si se utilizan con astucia.

Por desgracia, los límites ya mencionados previamente hacen que reconocer este tipo de elementos resulte complejo para la máquina, requiriendo un esfuerzo adicional frente a la rápida asociación que pudiera realizar un jugador humano al contemplarlos. La capacidad de adaptación también es limitada, pues es poco creíble que al enseñar a la IA a reconocer este tipo de patrones, se vayan a tener en cuenta todas las combinaciones que pueden llegar a darse en un mapa. Si hay alguna permutación que no se ha previsto, la máquina no sabrá reconocerla y, por tanto, no podrá beneficiarse de sus ventajas estratégicas.

Las unidades

En lo que respecta a los efectivos de los que dispone cada bando, se pueden observar dos dimensiones diferentes que condicionan la toma de decisiones: Las unidades aliadas y las unidades del bando enemigo. Y, lo que es más importante, la relación entre estos dos ámbitos. Una IA con un ejército que supere en potencia y número al de su adversario podría permitirse ciertas indulgencias y presionar más a costa de perder mayores efectivos, con tal de lograr capturar ciertos puntos importantes. Una IA con un ejército más reducido podría verse obligada a usar tácticas más prudentes y a proteger a sus unidades con mayor ahínco, buscando siempre una alternativa segura en sus estrategias.

La distribución inicial de las unidades también importa. ¿Están situadas en una formación compacta, pudiendo avanzar juntas en todo momento para apoyarse las unas a las otras, o se encuentran distribuidas por todo el mapa, debiendo invertir pues una serie de movimientos en reunirlas en distintos grupos para reforzar su posición? ¿Controlan puntos estratégicos de terreno, o se encuentran en una desventaja inicial respecto al adversario, que cuenta con un despliegue potencialmente más ventajoso?

Aunque a la larga siempre se puede medir la situación de las unidades sobre la marcha, adaptando la estrategia en consecuencia, un conocimiento previo podría permitir una planificación integrada en la IA que ofreciera una serie de directrices iniciales con las que obtener una ventaja rápida. Sin embargo, y al igual que con el terreno, esto iría contra la idea original del proyecto, que es simular un jugador autónomo capaz de combatir en cualquier clase de terreno.

9.3. Problemas de bajo nivel

A diferencia de los problemas de alto nivel que se han listado antes, los problemas de bajo nivel se orientan a resolver detalles más concretos, en muchas ocasiones relacionados con los problemas de alto nivel. Por tanto, se profundizará más en ellos, intentando especificar los posibles peligros potenciales a la hora de buscar una solución para ellos.

9.3.1. Tamaño del mapa

El mapa puede variar, como se ha especificado previamente, pero sin duda uno de los aspectos más importantes relacionados con este ámbito es el tamaño del mapa completo, que puede variar según el nivel. Por tanto, el reto podría pasar de tener que analizar una sencilla cuadrícula de 10x10 a tener que manejarse en un mapa de dimensiones mucho más amplias. Esto condicionará a la IA de forma severa.

Mapas pequeños y mapas grandes

Para entender mejor el reto que supone un tamaño variable, se pasará a comprobar las posibles ventajas de un mapa pequeño frente a las características de un mapa grande. Qué se entiende por

pequeño y grande es relativo, pero se puede realizar una breve aproximación mental para teorizar al respecto sin entrar en tecnicismos.

Los mapas pequeños son fácilmente analizables. Una unidad podría comprobar cada una de sus posibilidades de movimiento en un mapa de 10x10 sin conllevar por ello un coste computacional alto. Esto se debe a que, habiendo 100 posibles cuadrículas en el mapa (y descartando aquellas que ya se encuentran ocupadas por otras unidades, aquellas que no son accesibles o a las que la unidad no puede llegar en ese turno), el orden del algoritmo que use dicha unidad será, probablemente, algo asumible en un periodo de tiempo aceptable.

La misma situación, sin embargo, en un mapa de gran tamaño, daría lugar a un incremento exponencial del coste computacional hasta hacerlo inviable. Calcular el impacto que el movimiento de dicha unidad pudiera tener en un mapa de 100x100, por poner un ejemplo, conllevaría probablemente minutos de análisis, dependiendo de cómo se haya implementado dicho análisis.

La forma del mapa

Pero el tamaño no es lo único que importa, sino también la forma. Lo más sencillo podría parecer, en un primer momento, el cuadrado: Corresponde con la representación mental de una matriz, y facilita al programador las opciones a la hora de implementar los distintos algoritmos de la IA. Sin embargo, en algunas ocasiones se podría buscar un mapa con unos límites distintos, definidos de forma arbitraria.

¿Cómo podría introducirse esa información en las rígidas estructuras de datos que proporciona un ordenador sin necesidad de derrochar tiempo y memoria a la hora de acceder a ellas? No es del todo coherente con la idea de ofrecer libertad de opciones, al fin y al cabo, el obligar a la persona que vaya a crear un nuevo nivel a ceñirse a formas tan cerradas.

9.3.2. Caminos

Siguiendo con la idea del mapa, otra cosa que probablemente entrañe cierta dificultad a la hora de la implementación es la búsqueda de caminos entre dos puntos concretos de dicho mapa.

Distancias relativas

Los distintos tipos de terreno que pueden darse en el mapa poseen características únicas, que los diferencian entre sí y sientan las reglas mediante las cuales las distintas unidades pueden interactuar con ellos. Una unidad podrá cruzar un tipo de terreno con pasmosa facilidad, pero quedar retrasada de forma crítica en otro adyacente de un tipo distinto. Esto implica que, al a hora de implementar las funciones de análisis del terreno, se deba tener en cuenta que la distancia entre dos puntos no siempre es el número de casillas que se necesita recorrer entre dichos puntos.

Por el contrario, se podría llegar a afirmar que, en muchos casos, acceder a una casilla cercana podría llegar a ser algo costoso e incluso peligroso usando la distancia más corta aparente (la recta entre los dos puntos), mientras que tomar un camino más largo que dé un rodeo podría suponer que el coste descendiera de forma dramática.

Un requisito esencial

La importancia de este problema radica en el que, probablemente, el algoritmo utilizado para buscar el camino entre dos casillas, y sus posibles variantes, será utilizado por toda clase de procesos y abstracciones de mayor nivel, que esperarán el resultado de dicho algoritmo lo más rápido posible para poder tomar decisiones al respecto. De hecho, a la hora de analizar distintas alternativas para

una unidad, es casi seguro que se necesitará hacer uso de él en un alto número de ocasiones, por lo que la eficiencia temporal del sistema de búsqueda debe de ser la mayor posible para no ejercer un efecto de bola de nieve: Agrandar aún más el alto coste temporal de otros algoritmos de alto nivel.

Siempre cabe la posibilidad, por otro lado, de realizar un proceso exhaustivo de análisis previo a la partida, para establecer algunas ayudas que luego puedan simplificar el proceso, pero en cualquier caso, la constante sucesión de movimientos de unidades y la alta cantidad de combinaciones posibles que pueden darse a la hora de trazar caminos hace que, sin duda alguna, este sea uno de los problemas de bajo nivel más críticos.

Por suerte, la búsqueda de caminos es un ámbito bastante estudiado en el campo de la inteligencia artificial, y no hay necesidad de ïnventar la rueda"de nuevo. Hay una gran cantidad de ejemplos de algoritmos y soluciones genéricas que se pueden utilizar para enfocar el problema desde una nueva perspectiva, adaptando dichas soluciones a los requisitos específicos de este.

9.3.3. Heurística

La definición de la heurística que se utilizará para comparar las distintas opciones es importante, y debe de estar lo suficientemente detallada para que no dé lugar a un diseño pobre y poco eficiente. Al fin y al cabo, es justamente dicha heurística la que decidirá qué alternativas se toman y, por tanto, la efectividad de la estrategia final.

El componente arbitrario

Como cabe esperar, estas directrices son profundamente subjetivas, y deberán ser implementadas mediante constantes y procesos específicamente orientados a evaluar componentes concretos, decididos por el diseñador. Al fin y al cabo, es la heurística utilizada la que decide si, a la larga, un algoritmo planteado de forma correcta genera los resultados esperados.

Requiere también, por tanto, un proceso exhaustivo de ensayo y error, para decidir cuáles de estas constantes son realmente necesarias y, una vez introducidas, el valor que deben tener. Una heurística mal planteada puede hacer que una propuesta interesante y bien encaminada obtenga unos resultados desastrosos, e incluso puede llevar al diseñador a descartar dicha propuesta, que de otra manera habría resultado sorprendentemente efectiva.

La naturaleza de la heurística

La heurística debe de estar bien definida, y ser medible. Para comparar se necesita dejar poco espacio para la interpretación, y al trabajar la máquina con absolutos, es natural pensar en los valores numéricos como principal elemento decisivo. Sin embargo, la composición de este conjunto de valores numéricos puede variar. ¿Se guardará la heurística en una única variable que poder comparar con otras? ¿Dependerá de varios valores numéricos, que actúen como pesos ponderados a la hora de tomar una decisión u otra? ¿Podrán dos conjuntos de valores heurísticos dar lugar a elegir una opción en un estado, y otra distinta en el otro?

Aunque la solución más obvia parece la simplicidad, a la hora de plantear una estrategia adecuada para combatir en el juego se encuentra que esta aparece llena de matices y distintos valores relativos que pueden pertenecer a diversos ámbitos, pero influir de forma similar en el resultado final. Hay que encontrar, por tanto, un compromiso entre complejidad y simplicidad, para hacer de la heurística algo claro y medible sin por ello obviar la naturaleza compuesta de la estrategia militar.

9.3.4. Estructuras subyacentes

Una de las prestaciones que ofrece Game Maker, el entorno utilizado, es la de poder acceder a los distintos elementos del juego de forma directa desde el código, buscándolos si así es preciso en la propia ventana del juego y en tiempo real. Esto, sin embargo, es un arma de doble filo: Puede resultar cómoda en muchos casos, pero conlleva un coste de computación alto, que no es aceptable en algunos algoritmos con una fuerte carga de trabajo.

El esqueleto invisible

Por ello se deduce que, al buscar la rapidez en los procesos más tediosos, es importante contar con ciertas estructuras .ºcultas"que faciliten el trabajo de la IA. Estas estructuras pueden contener referencias a los objetos y pueden ser modificadas cuando estos cambien su estado o posición, actuando como una referencia constante que el jugador no puede ver pero que descarga el trabajo de las distintas funciones que se ejecutan en segundo plano mientras este juega.

El más claro ejemplo de estructura es una matriz que corresponda con el mapa de juego, y que guarde unidades y terrenos en las casillas que les corresponderían de ser dicho mapa una representación gráfica de esa matriz. Estas estructuras pueden llegar a ser muy útiles, pero tienen una pega: Deben de ser actualizadas en todo momento de manera que no haya discrepancias entre lo que se representa en pantalla y los datos que contienen.

9.3.5. Variedad de opciones

El último problema de bajo nivel que se nos presenta es la variedad a la que se está aspirando. Es indudable que una IA orientada a un juego con un único tipo de unidad y dos o tres tipos de terreno sería mucho más sencilla que la que se está planteando en este proyecto. Al contar con menos opciones y menos estados posibles, podría centrarse en estrategias más complejas dentro del abanico que se le ofrece.

Al añadir más tipos de unidades, más acciones y más características especiales, se ofrece más variedad al usuario y se enriquecen las partidas, pero también se entorpece a la IA, que debe pasar de trazar estrategias con un conjunto de posibilidades relativamente limitado a adaptarse a un nuevo repertorio de nuevas combinaciones.

9.4. Un primer esbozo

Para concluir con el capítulo, se ofrece un breve esbozo de lo que podrían ser las soluciones a los problemas antes mencionados. Aunque la solución completa, así como un estudio más intensivo de los distintos algoritmos conocidos que podrían ser utilizados en estos casos, se mostrarán más adelante, conviene realizar un primer acercamiento para irse formando una idea aproximada de las características de la IA.

9.4.1. Dirigir un ejército

Para poder dirigir su ejército de forma eficiente, la IA deberá plantear soluciones inmediatas y a largo plazo para acabar con el enemigo, y mover sus unidades de forma consecuente. Esto es más sencillo en concepto que en la práctica, pero se intentará tener presente la estrategia general a la hora de diseñar la táctica individual de cada unidad.

9.4.2. Acciones

Para evaluar las distintas acciones que puede realizar cada unidad, se establecerán una serie de parámetros con los que poder valorar lo deseable de dicha acción. Estos parámetros están estrechamente relacionados con el problema de bajo nivel relacionado con la heurística.

9.4.3. Recursos limitados

Se establecerán límites arbitrarios a la hora de analizar opciones y ejecutar las acciones, de tal manera que el sistema no se sature. Esto volverá a la IA algo menos eficiente, pero permitirá que esta trabaje dentro de los límites de tiempo establecidos.

9.4.4. Cooperación

La cooperación entre muchas unidades es compleja, por lo que se ha decidido dividir al ejército en distintos grupos de unidades que puedan ser manejables y actuar en conjunto para obtener un bien común, sin tener en cuenta a otros grupos del mismo bando para evitar la saturación de recursos.

9.4.5. Solapamiento de órdenes

Para evitar que las unidades se entorpezcan y que haya errores, estas actuarán siempre en orden riguroso, intentando dar prioridad a aquellas con mejores expectativas. A la larga, es inevitable que se den combinaciones poco efectivas, pero se intentará paliar este efecto definiendo claramente los efectos deseados y las unidades que más fácilmente pueden obtenerlos, dejando que sean estas las primeras en actuar y haciendo que el resto se adapte a las consecuencias.

9.4.6. El coordinador

Se podría dar un valor de prioridad a las unidades que las haga más propensas a ser el coordinador, pudiendo dirigir por tanto a otras unidades para favorecer sus acciones, que al ser de las más relevantes de su grupo normalmente tendrán un impacto mayor en el juego. La figura del coordinador, sin embargo, irá cambiando de una unidad a otra según se requiera, y cualquier unidad podrá ser coordinadora llegado el momento si se dan las condiciones necesarias.

9.4.7. Un entorno variable

La única solución a este problema es hacer la IA lo más adaptable posible, de tal manera que pueda trabajar de forma adecuada en entornos distintos a costa de perder eficiencia frente a otras IA especialmente diseñadas para un mapa concreto.

9.4.8. Tamaño del mapa

Para evitar la saturación de recursos temporales y físicos en mapas grandes, se ha decidido implementar una "niebla de guerra", un concepto que implica que, a la hora de tomar decisiones, el conocimiento del entorno de una unidad se ve limitado en el espacio a lo que puede "ver". Este campo de visión vendrá dado por el coordinador de cada grupo específico de unidades, de tal manera que dichas unidades podrán realizar sus acciones dentro de un sector controlado del mapa, sacrificando el conocimiento de otras zonas más lejanas a cambio de poder realizar un análisis más intensivo de lo que tienen cerca.

La forma del mapa se ha establecido como cuadrada para simplificar las estructuras de datos relacionadas con este. Sin embargo, se plantea la posibilidad de realizar mapas con formas distintas añadiendo terrenos de tipo infranqueable en los límites del mapa, y definiendo por tanto manualmente la zona a la que se puede acceder.

9.4.9. Caminos

Se analizarán varios algoritmos de búsqueda de caminos para encontrar uno eficiente y adecuado a las necesidades de la IA.

9.4.10. Heurística

Se ha decidido dejar la heurística como un único valor numérico que, sin embargo, estará influenciado por varios factores, pudiendo variar de manera drástica según las distintas condiciones y la manera que tenga de valorarlas cada aspecto de la IA.

9.4.11. Estructuras subyacentes

Se ha decidido implementar dos estructuras principales: Una para tener un acceso rápido a los terrenos y otra para acceder a las unidades. Estas estructuras se corresponden con el mapa del juego, y tienen una utilidad meramente funcional, para ahorrar tiempo. Se plantea la posibilidad de una tercera estructura, generada al ser creado el nivel, que recoja ciertos puntos guía que las unidades puedan usar para orientarse en mapas de gran tamaño.

9.4.12. Variedad de opciones

Se ha decidido mantener la IA como algo genérico y no orientado a cada tipo de unidad, de modo que las trate a todas de forma similar, sin hacer excepciones. De esta manera, se pierde cierta eficiencia pero se evita que la introducción de una nueva unidad (algo que no está descartado) desequilibre el sistema.

Capítulo 10

Estudio de algoritmos

10.1. La búsqueda del algoritmo adecuado

La diferencia entre una IA bien planificada y otra caótica se debe, en cierta medida, al trabajo de investigación previo al diseño de dicha IA. La idea de ponerse "manos a la obra"normalmente resulta atractiva, pero es habitual encontrar que, con un breve trabajo de investigación previo, se pueden encontrar soluciones ya formuladas que ofrecen una base sobre la que poder trabajar y obtener, a la larga, un producto más refinado y complejo de lo que se podría haber logrado empezando desde cero.

Sin embargo, no todos los algoritmos son adecuados para todo tipo de trabajo. Se debe pensar en un algoritmo como en una herramienta: Cada uno sirve para una función específica, aunque con algunas modificaciones pueden ser adaptados a distintas tareas. Primero se debe plantear lo que se necesita, y luego investigar al respecto, eligiendo solo aquello que tenga relación con el problema que se va a afrontar.

10.2. Algo de historia

Para entender los campos de estudio más conocidos en relación a lo que se pretende en este proyecto, conviene primero adquirir unas nociones generales sobre sus orígenes.

10.2.1. Teoría de juegos

La inteligencia artificial, y en concreto la teoría de juegos y los problemas de análisis y toma de decisiones, no es algo nuevo. La teoría de juegos se empezó a formar al tratar de estudiar diversos juegos de cartas y estrategia [1]. Incluso después de abandonar dicho ámbito, se conservó en nombre, aunque hoy en día se suele atribuir, sobre todo, al ámbito económico. Algunas de las personalidades que hicieron de este un campo de estudio son, por citar dos ejemplos, John Von Neumann [23] y John Forbes Nash [21].

Se podría definir la teoría de juegos como la aproximación matemática a las situaciones conflictivas. Estas situaciones pueden contener varios agentes que toman decisiones independientemente, generando entre todos un resultado común. Este resultado puede ser más o menos satisfactorio para cada agente, ya que estos tienen sus propias preferencias.

Las situaciones que trata la teoría de juegos se pueden clasificar en varias categorías: cooperativos o no cooperativos, simétricos, asimétricos, de suma cero o suma no nula, de estrategias simples o reactivas... [12] En cualquier caso, las aplicaciones que suelen tener los problemas relacionados con la teoría de juegos en la informática suelen estar relacionadas con la semántica y los distintos algoritmos de toma de decisiones en los que se apoyan los distintos planteamientos de esta.

De forma más específica, podemos usar algunos de los conceptos que se presentan en la teoría de juegos para analizar la estrategia que se desea implementar en la IA. Al fin y al cabo, se puede considerar que el juego que se está implementando comparte algunas de las características propias de esta teoría: Dos agentes con preferencias personales que influyen en un terreno común y cuentan con una estrategia concreta.

10.2.2. Algoritmos de búsqueda

La teoría de grafos ha estado presente en el ámbito científico desde el siglo XVIII, si bien se considera que el establecimiento de la teoría y el glosario relacionado se dio en 1976, por los matemáticos Kenneth Appel y Wolfgang Haken [25].

Los algoritmos relacionados con esta teoría se podrían definir, de forma algo general, como soluciones a problemas relacionados con las conexiones entre distintos puntos destacados. Estos problemas siempre se pueden reducir a un grafo, esto es, representaciones visuales de dichos problemas usando puntos para representar los distintos elementos y líneas para las conexiones entre esos puntos.

La teoría de gráficos está directamente relacionada con la búsqueda del camino más eficiente entre dos puntos, por lo que merece la pena estudiar los distintos algoritmos que presenta para construir un sistema de caminos eficiente entre los terrenos del mapa.

10.3. Algoritmos específicos

Para reducir el ámbito de estudio a unas zonas concretas de estas teorías, primero hay que especificar lo que se busca. Por tanto, es indispensable analizar los problemas planteados en capítulos anteriores, para hallar aquellos que puedan estar relacionados con estos algoritmos.

Se analizarán únicamente aquellos algoritmos encontrados que aporten algo a la solución final de la IA.

10.3.1. Toma de decisiones

Cualquier algoritmo que ofrezca un planteamiento enfocado a la toma de decisiones puede servir como inspiración para diseñar la propia IA, ya que esta se verá obligada a decidir entre múltiples opciones constantemente, intentando maximizarlas contra el oponente. La estrategia general, las acciones de cada unidad o incluso la forma de abarcar los distintos desafíos que se le van presentado son ejemplos de situaciones en las que un algoritmo de la teoría de juegos podría ser útil.

10.3.2. Búsqueda de caminos

Uno de los principales problemas mencionados en capítulos anteriores es la dificultad de establecer caminos en un mapa variable y con terrenos diferenciados. Es indispensable pues aplicar un algoritmo eficiente de búsqueda de caminos, para evitar la carga del sistema.

10.4. Minimax

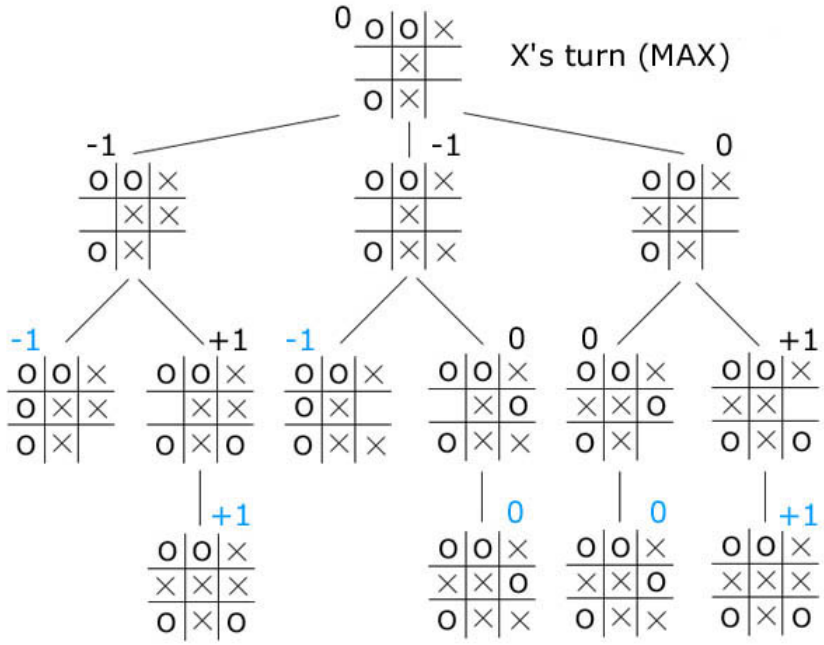


Figura 10.1: Ejemplo del árbol creado en Minimax

10.4.1. Planteamiento

El algoritmo Minimax es uno de los principales en la teoría de juegos. Está orientado a situaciones en las que dos agentes se turnan para tomar decisiones, y su filosofía se basa en la idea de que el agente rival siempre tomará la decisión más perjudicial (de ahí el nombre, pues se va alternando entre el valor máximo, que corresponde al turno del jugador, frente al valor mínimo cuando el turno es el del rival).

10.4.2. Explicación

Se plantea un juego en el que participan dos agentes independientes y enfrentados. En este juego, los dos agentes actúan por turnos, alternándose para elegir una acción que les proporciona un beneficio inmediato. En su turno, cualquiera de los dos agentes cuenta con una serie de opciones que puede utilizar, cada una con un valor asociado que representa el beneficio de escoger dicha acción.

El algoritmo se basa en que, en su turno, el jugador escogerá la opción que más le convenga, esto es, la que le reporte mayor beneficio. A su vez, el rival escogerá la que mayor beneficio le aporte a él, que por tanto tendría un valor mínimo para el jugador. De esta manera, se crea un árbol de decisión en el que se sobreentiende que en cada nivel se alternará entre tomar la opción con mayor beneficio o la que tenga el valor más bajo. La toma de decisiones se hará en base a este árbol [8].

En el ejemplo de la figura 10.1, se puede ver el algoritmo aplicado a una partida del conocido juego de las tres en raya. El objetivo del juego es colocar tres piezas en línea en un tablero de 3x3, alternándose los jugadores para colocar una pieza cada turno. En el árbol generado, se asigna un

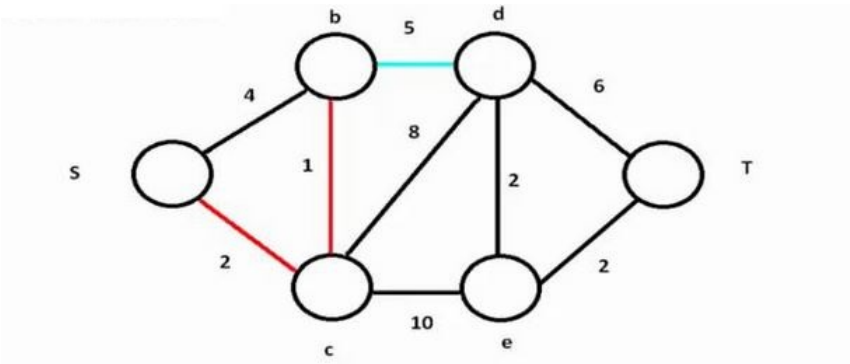
valor de -1 si la situación implica una derrota segura, de +1 si conlleva una victoria, y 0 en cualquier otro caso. Jugando en el turno de la X y usando el árbol como referencia, la opción elegida sería la tercera empezando por la izquierda. Es la única que no asegura una derrota segura si el rival elige el valor mínimo.

10.4.3. Aplicaciones

La aplicación de este algoritmo en la partida es bastante directa. Al fin y al cabo, el juego reúne varias de las características relacionadas con el algoritmo: Dos jugadores enfrentados, que se alternan para realizar acciones. Dichas acciones, además, buscan maximizar los beneficios de dicho jugador y minimizar los del rival. Por tanto, la secuencia es idéntica a la del algoritmo.

La gran cantidad de alternativas que ofrece el juego hace que el análisis exhaustivo que realiza Minimax para crear el árbol sea imposible. Además, el conocimiento que tienen los dos jugadores sobre el estado del juego es incompleto, ya que el azar juega un papel importante en el desarrollo de las acciones de cada unidad. A pesar de estas pegas, se puede extraer la filosofía del algoritmo para aplicarla, a grandes rasgos, en la propia IA, suponiendo que el rival actuará siempre de la forma más perjudicial posible para los objetivos del jugador e intentando anticiparse a esos movimientos.

10.5. Dijkstra



Vértice	Peso 1	Peso 2	Peso 3	Peso4	Peso5	Peso 6
S	(0,S)	*	*	*	*	*
B	(4,S)	(3,C)	(3,C)	*	*	*
C	(2,S)	(2,S)	*	*	*	*
D	-	(10,C)	(8,B)			
E	-	(12,C)				
T	-	-				

Figura 10.2: Ejemplo del algoritmo de Dijkstra aplicado a un grafo

Este algoritmo, cuyo nombre viene dado por su descubridor, Edsger Dijkstra, que lo propuso en el año 1959. También conocido como algoritmo de caminos mínimos, se centra en encontrar las distancias mínimas en un grafo tomando uno de los vértices como referencia de inicio. Es uno de los algoritmos más básicos de la teoría de grafos.

10.5.1. Planteamiento

Se cuenta con un grafo no direccionado G (esto es, las conexiones entre vértices son bidireccionales) con N vértices y una serie de conexiones entre vértices que tienen asignado un coste c, el cual varía dependiendo de los dos vértices a los que conecte. Se quiere encontrar, tomando uno de los vértices como inicio, el camino con el mínimo coste acumulado entre dos puntos (es decir, la suma de los costes de las todas las conexiones que recorre el camino que conecta el inicio y el final es la menor posible). El algoritmo de Dijkstra es un tipo de algoritmo voraz [5] que permite encontrar siempre la solución óptima a este problema, si es que la hay.

10.5.2. Explicación

El algoritmo de Dijkstra sigue un procedimiento progresivo, analizando cada vértice por separado en cada paso y las conexiones de dicho vértice hacia sus vecinos. Además, es un algoritmo con memoria, esto es, guarda los valores encontrados de tal manera que estos puedan ser comparados en pasos posteriores. El algoritmo empieza en el vértice de inicio.

En cada paso, elige el vértice con menor coste acumulado hasta el momento que aún no se haya elegido (el primer vértice tiene un coste de 0) y calcula el coste total hacia sus vecinos, sumando dicho coste acumulado al de la conexión con cada uno de ellos. Si es la primera vez que se analiza dicho vecino, el algoritmo le asigna dicho coste acumulado de forma automática, pero si ya se ha analizado antes desde otro vértice, solo se lo asigna si es menor al valor que se introdujo para dicho vértice la última vez. En cualquiera de los dos casos, también le asigna el camino completo para llegar hasta dicho vértice.

Una vez se han analizado los distintos vecinos, se pasa a escoger aquel con el menor coste acumulado que todavía no se haya analizado. De esta manera, se evita analizar dos veces el mismo vértice. En el momento en el que se analice el vértice de destino, se dará el algoritmo por finalizado y se ofrecerá el camino más corto encontrado.

10.5.3. Aplicaciones

El tipo de grafo propuesto en el algoritmo de Dijkstra puede funcionar como abstracción para el propio mapa del juego, por lo que dicho algoritmo resulta especialmente útil. Además, la necesidad de encontrar el camino más corto entre dos puntos será un requisito constante sobre el que los distintos procedimientos de alto nivel, como por ejemplo los de elección de órdenes, se apoyarán una y otra vez. La utilidad de Dijkstra es, por tanto, bastante alta en este sentido.

También resulta interesante recordar la capacidad de memoria de Dijkstra para otro tipo de algoritmos, como el que se aplicaría al intentar encontrar todas las unidades del bando rival amenazadas por una unidad. Esto implicaría encontrar el camino más corto entre dicha unidad y cada una de aquellas a las que amenaza, porque de un modo simplificado podría verse como varios algoritmos de Dijkstra aplicados de forma secuencial. Sin embargo, la herramienta más útil que propone en este caso el algoritmo es la matriz de costes acumulados, que permite saber el coste mínimo alcanzado para llegar a cada vértice, y que por tanto evita que ese vértice sea analizado en dos ocasiones distintas si se sigue un sistema de búsqueda similar al de Dijkstra.

10.6. A estrella

El algoritmo A* es un algoritmo de búsqueda ampliamente utilizado en videojuegos y en teoría de grafos, aunque sus aplicaciones se extienden también a otros ámbitos en los que se pueda extrapolar el problema a un grafo, como la búsqueda del camino más corto en mapas.

10.6.1. Planteamiento

Se cuenta con un grafo en el que se quiere llegar desde un vértice a A a un vértice B eligiendo el camino con el menor coste posible. El conocimiento del entorno es completo y, por tanto, se puede acceder en todo momento a las características concretas de cada vértice o conexión. El algoritmo siempre encuentra una solución, si esta existe, y ofrece una mayor eficiencia frente a otros algoritmos de búsqueda exhaustiva.

10.6.2. Explicación

A estrella (o asterisco, como se le denomina en ocasiones) se basa en la utilización de un criterio definido de antemano que permita evaluar lo prometedora que es un alternativa. Uniendo esta valoración a la del coste acumulado para llegar a dicha alternativa se obtiene una heurística que permite comparar las distintas alternativas, recorriéndolas de modo que siempre se elijan aquellas con mejores perspectivas.

A diferencia de otros algoritmos que no usan este tipo de heurística, A* evalúa algo más que el coste bruto, y por tanto reduce su tiempo de computación al no tener que recorrer vértices poco prometedores. Sería el caso de un algoritmo que diera prioridad a los puntos del mapa más cercanos, en una línea recta, al destino, frente a ellos que a pesar de tener un coste bajo están más alejados. Esto no implica que dichos puntos no puedan ser parte de la solución final, pero sin duda esto será menos frecuente.

La eficiencia de A* radica, pues, en la implementación de una heurística eficiente que permita valorar con un grado suficiente de certidumbre lo prometedora que es una alternativa. En el caso de mapas en los que el coste de moverse de un punto a otro varía dependiendo del terreno, la primera idea que viene a la cabeza, como se ha mencionado antes, es la distancia neta entre dichos dos puntos sin tener en cuenta el coste, es decir, la medida de una línea recta que los una. Sin embargo, está es solo una de los muchos tipos de heurística que pueden aplicarse, cada uno de los cuales dependerá de las características del problema que va a resolverse.

10.6.3. Aplicaciones

La aplicación de A* resulta obvia en este caso, ya que en el mapa que se plantea es habitual que el mejor camino esté ubicado alrededor de la línea recta que une a dos unidades. Aunque esto puede cambiar dependiendo de los tipos de terreno que haya entre dichas unidades, la heurística podría servir para reducir considerablemente el tiempo de computación necesario para hallar la distancia mínima entre dos puntos, añadiendo pues A* una ventaja sobre Dijkstra que radica en la introducción de dicha heurística.

Capítulo 11

Especificaciones de la IA

11.1. El diseño final

En este capítulo se presenta el resultado final de la IA, una vez realizada su implementación, así como una breve visión de los escollos encontrados durante dicha implementación.

11.1.1. Consideraciones previas

La IA que se ha implementado es una propuesta a los requisitos y problemas listados en capítulos anteriores. No es, por otro lado, una solución óptima al problema, sino más bien un acercamiento a una serie de ideas que pueden ejercer un buen funcionamiento en la práctica. En los próximos capítulos se realizará una evaluación final de los resultados, así como una propuesta de posibles mejoras que podrían aplicarse para incrementar el rendimiento de esta.

Se debe tener también en cuenta de que, a pesar de haber diseñado la IA de la forma más genérica posible, hay ciertos datos introducidos de forma arbitraria en el diseño, como podría ser la prioridad de unidades, que parecen chocar con la idea de ofrecer un sistema versátil y polifacético. Es importante hacer notar la necesidad de esta adaptación para lograr un sistema de IA que pueda ofrecer cierta competencia, pues uno excesivamente genérico podría dar lugar a una toma ineficiente de decisiones por parte de la máquina. Para mantenerse fiel a la intención original del proyecto, se ha reducido este tipo de ajustes arbitrarios al mínimo, obteniendo un compromiso de eficiencia sin por ello perder su esencia original.

11.1.2. Aspectos generales

La IA tiene en cuenta varios puntos que son especificaciones esenciales del videojuego en el que debe actuar, y actúan como reglas inamovibles que deberán explotarse al máximo para obtener un resultado favorable al diseñar una estrategia de juego. Algunas de las siguientes reglas han sido especialmente valoradas durante la implementación, adaptando la IA a ellas de tal forma que estas actúen como ventajas estratégicas.

Variedad de unidades

El amplio abanico de unidades presentes en el juego ofrece una gran cantidad de opciones a la hora de definir la actitud de la IA en lo referente a cada tipo de unidad. Sin embargo, se ha decidido

centrarse en los atributos comunes a todas ellas, creando reglas absolutas por las que se decide el uso que se va a dar a cada una, en vez de tratar cada tipo de unidad de una manera distinta.

Esta forma de tratar a las unidades resulta algo menos eficiente, ya que evita tácticas predefinidas que puedan ofrecer cierta ventaja en tiempo y eficiencia frente al análisis exhaustivo que se ha decidido escoger. Sin embargo, también evita utilizar estrategias poco efectivas ante nuevos tipos de unidades con características especiales.

Por ejemplo, sería totalmente aceptable diseñar una estrategia mediante la cual las unidades que pueden atacar a cierto rango de distancia deban de colocarse preferentemente, y si es posible, tras otras unidades de combate directo, para poder estar protegidas frente a otras unidades enemigas sin perder por ello su capacidad de atacar. Ahora, se podría plantear también la introducción de una unidad cuya efectividad en el ataque fuera anecdótica al atacar desde lejos, pero aumentara de forma significativa al encontrarse en una posición adyacente a otra. La aplicación de la estrategia mencionada con dicha unidad daría lugar a un comportamiento menos eficiente de lo esperado, que en muchos casos podría llegar incluso a resultar perjudicial para el propio bando de la unidad.

Como es imposible prever las características específicas que pudieran tener las nuevas unidades introducidas por un hipotético programador que aplicara sus modificaciones al videojuego, se debe intentar plantear un sistema que, de forma genérica, ofrezca siempre que sea posible soluciones adecuadas, obteniendo un compromiso entre versatilidad y efectividad independientemente de los elementos con los que deba tratar.

De ahí que se utilicen los atributos antes mencionados, comunes a todas las unidades, como vida, efectividad de ataque o número de acciones, como indicadores a valorar para decidir qué uso puede tener dicha unidad. Esto indica que, en diferentes situaciones, un mismo tipo de unidad podría ejercer dos roles completamente distintos.

Múltiples órdenes

Un aspecto fundamental del juego es el coste de acciones que conlleva el ejecutar una orden. Al realizar un movimiento, ataque o cualquier otro tipo de resolución, la unidad resta de su total de acciones restantes para ese turno una cantidad arbitraria asociada a ese tipo de acción.

Esto ofrece una complicación adicional frente a un sistema de juego en el que una unidad pudiera realizar una única acción, tras lo cual pasaría a estar inactiva para el resto del turno. Aunque la implementación de la IA en un sistema así podría resultar más sencilla al no existir la posibilidad de usar una unidad en más de una ocasión, también cierra las puertas a posibles estrategias avanzadas en las que una misma unidad puede tener varias funciones distintas en un mismo turno, como atacar y retirarse o cubrir a otra unidad y establecer una guardia en su terreno.

El potencial que nace de la posibilidad de obtener distintos beneficios en un mismo turno usando para ello un único elemento hace que la cooperación entre unidades sea mayor. La IA propuesta abraza el concepto de reutilización de unidades, permitiendo repetir el proceso de toma de decisiones para una misma unidad en varias iteraciones de un mismo turno, eligiendo cada vez una opción diferente (o similar a las anteriores) siempre y cuando la unidad cuente con suficientes acciones para realizar dichas opciones.

Consecuencias de las acciones

El hecho de realizar acciones al mismo tiempo puede dar lugar a errores y a una reducción significativa de la ventaja táctica que ofrecen las acciones de las unidades implicadas, si no se gestiona de forma correcta. Para evitar complicar demasiado el nivel de toma de decisiones en este aspecto, se ha decidido que las acciones de cada unidad bloqueen temporalmente el sistema de toma de decisiones hasta que la acción haya terminado y las consecuencias se hayan calculado y aplicado.

De esta manera, se sigue un aproximamiento que recuerda a los algoritmos voraces, en el sentido de que no hay "vuelta atrás": Una vez se ejecuta una acción, esta no puede ser deshecha, y los futuros análisis que se hagan sobre el terreno y las unidades deberán de tener en cuenta que los resultados de las acciones previas están ya presentes en el juego.

Por tanto, es importante planificar bien cada acción, eligiendo aquella que sea más deseable entre las que se han podido encontrar, e intentando reducir al máximo los perjuicios que puedan obtenerse al realizar dicha acción.

Dividir para vencer

Al no saber cuál será el tamaño del ejército aliado, hacer un recorrido exhaustivo por el árbol de alternativas puede tener un coste variable, en algunas ocasiones intolerable para las capacidades del sistema. Por ello se propone una división del problema, en lo referente a las unidades a un nivel más general, y en los aspectos que influyen en la toma de decisiones a un nivel algo más abstracto.

Este proceso está relacionado estrechamente con la filosofía del "Divide y Vencerás", que se centra en seccionar un problema de gran envergadura en distintos problemas más pequeños, que al ser resueltos de forma individual pueden combinarse para obtener una solución global satisfactoria que no podría haberse conseguido tratando dichos elementos como un todo. Al dividir a las unidades, se consigue crear grupos reducidos de trabajo en los cuales la carga computacional es siempre soportable por el sistema. Al dividir los aspectos que influyen en la toma de decisiones se evita una elección aleatoria de alternativas a la hora de comprobar qué órdenes puede realizar la unidad, reduciendo el conjunto de opciones de forma que solo aquellas más prometedoras tengan cabida en la valoración.

Sin embargo, no hay que olvidar que las unidades siguen actuando como un conjunto, y aunque sería extremadamente difícil coordinar grandes grupos de unidades, sí se debe intentar que los pequeños grupos que se establecen actúen en conjunto, pudiendo obtener resultados comunes que, sin llegar a ser tan efectivos como una estrategia global, sí pueden ofrecer la ilusión de una mente táctica tras el jugador máquina.

Entorno imprevisible

De nuevo se debe recalcar la naturaleza versátil de la IA, pues el entorno al que debe hacer frente cambiará en cada turno, y en cada partida. Se ha estudiado la ejecución del sistema de IA en distintos niveles predefinidos, cambiando los valores y las directrices constantes para evaluar los resultados, pero a la larga se ha optado por conservar algunos factores especialmente genéricos para intentar anticiparse a la gran variedad de situaciones que se pueden dar.

11.2. Algoritmos de bajo nivel

Algunos de los algoritmos de bajo nivel que se utilizan en la IA sirven para automatizar procesos que, de otra manera, serían especialmente costosos. Aunque el principal atractivo del trabajo que se está realizando es el flujo de la IA a un nivel más general, es interesante evaluar estos pequeños procesos para entender mejor la efectividad general.

11.2.1. Versiones de A*

Como ya se ha mencionado en ocasiones anteriores, el algoritmo A* resulta especialmente útil en esta clase de entorno, pues permite ahorrar el gasto de tiempo que se invierte en la búsqueda de

caminos. Sin embargo, el algoritmo implementado ha sido modificado ligeramente para adaptarse a las necesidades de cada búsqueda.

Adaptando A*

Para empezar a entender la adaptación que se ha hecho de este conocido algoritmo, se deben comprender antes las características del terreno en el que se debe operar. La cuadrícula del mapa implica que cada terreno (vértice) se encuentra unido a sus vecinos en dirección ortogonal (es decir, todos aquellos terrenos que compartan al menos un lado con el cuadrado que representa al terreno actual). Las diagonales se han obviado, por lo que queda un total de cuatro vecinos para cada terreno, exceptuando aquellos que se encuentran en el límite del mapa. Además, las conexiones entre los vértices tienen un coste definido por el coste de movimiento a dicho vecino, teniendo en cuenta el tipo de la unidad.

Este último detalle es especialmente relevante, pues de tener un coste fijo se podría realizar un trabajo previo (o ir guardando las distintas búsquedas realizadas) para simplificar el trabajo posterior y obtener una estructura de datos de soporte. Sin embargo, el hecho de que cada unidad tenga un coste distinto para acceder al mismo tipo de terreno hacen que esta estructura pierda sentido, haciendo preferible la búsqueda dinámica usando A*.

Al definir la heurística que se utilizará en el algoritmo, se ha pensado en la distancia más corta entre dos terrenos, siguiendo las reglas de movimiento (es decir, no se puede salir de un terreno si no se accede a uno de sus cuatro vecinos) y considerando que el coste de movimiento de un terreno a otro es constante e igual a 1. Esto coincide con la distancia Manhattan, y ofrece una aproximación inexacta aunque útil de la cercanía con el punto de destino. Al fin y al cabo, es más probable que una casilla con menor heurística que otra obtenga un camino más corto a su vez.

Esto último no ocurre siempre, y la aparición de terrenos difíciles como montañas, zonas desérticas o incluso terrenos infranqueables hace que, en muchas ocasiones, dar un gran rodeo ofrezca una reducción en el coste de movimiento significativa. Por ello se proporcionan distintas estructuras de datos para ir guardando las diferentes alternativas que se vayan presentando. La más significativa es una cola de prioridad, que permite elegir siempre aquella alternativa con mejor heurística simplificando a su vez la introducción de nuevas alternativas en el orden correcto.

Versiones de A*

Además de la versión básica (que puede tener o no solución dependiendo de si la unidad es capaz de llegar al destino con su movimiento) se han implementado distintas alternativas para su uso en algunos algoritmos específicos:

A* en rango

En esta variante, se considera que se ha llegado al final del algoritmo cuando se accede a cualquiera de las casillas indicadas en un rango concreto de terreno, que puede contener uno o más puntos en el mapa. La heurística utilizada es la menor distancia hasta cualquiera de las casillas que componen dicho rango. Este algoritmo es especialmente útil a la hora de hallar si una unidad puede atacar a otra sin importar la localización desde donde lo haga, pues se tienen en cuenta todas las posibles opciones desde las cuales la unidad podría ejecutar dicho ataque.

A* infinito

En este algoritmo se plantea la posibilidad de contar con acciones ilimitadas para recorrer el terreno. Esto es especialmente peliagudo, pues si no se maneja correctamente puede darse lugar a una búsqueda infinita que no tiene solución. Por tanto, siempre que se invoca este algoritmo se hace planteando un límite arbitrario o sabiendo previamente que existe una solución.

A* infinito en rango

Este algoritmo es la conjunción de los dos anteriores.

A* esquivando

Este algoritmo es una alteración de A* infinito, que se centra en asegurar que la última casilla accesible (esto es, el último terreno al que la unidad puede acceder con sus acciones actuales) esté vacía. Sirve para encontrar el segmento de mayor tamaño al que se puede acceder con las acciones actuales del camino más corto a una casilla concreta del mapa.

A* esquivando en rango

Este algoritmo añade el concepto del rango de casillas de destino a A* esquivando.

A* atravesando

Este algoritmo se usa únicamente en la IA, y utiliza los mismos preceptos que el anterior, pero añade una nueva condición: Una vez dejado atrás el segmento al que se puede acceder en ese turno, las unidades enemigas no obstaculizan el movimiento a la hora de calcular el resto del camino. Esto es debido a que la disposición de dichas unidades puede cambiar en turnos posteriores.

11.2.2. Puntos guía

Cuando se trata de analizar mapas amplios, el coste de encontrar un camino entre dos puntos puede ser excesivo. Para ello, se ha decidido implantar una solución que opera con un tipo de elemento especial al que se ha nombrado como "punto guía". Estos puntos guía se distribuyen al principio de la batalla de forma ordenada en distintos puntos del mapa, separados entre sí de manera que cada uno çontroleüna zona determinada de terreno. Después de distribuirlos, se calculan los costes entre ellos y, por último, se asocia una matriz a cada uno en la que se muestra la dirección que se debe tomar para llegar a cualquier otro punto guía del mapa.

De esta manera, si una unidad quisiera alcanzar una posición que se encuentra a una distancia mayor de lo acostumbrado (esto se compararía con un valor arbitrario establecido sobre la experiencia), no lo haría calculando la distancia de manera habitual, pues se entiende que en ese mismo turno no será capaz de alcanzarlo. En vez de eso, buscaría el punto guía más cercano y comprobaría a qué otro punto guía adyacente debe intentar acceder para iniciar el camino hacia su destino, usando la matriz asociada a dicho punto guía.

Esto permite establecer patrones de movimiento a largo plazo, que permitan prever el recorrido que se va a realizar durante varios turnos para alcanzar una posición deseada. Sin embargo, también ofrece algunas desventajas.

1. Puntos guía estáticos: Los puntos guía nunca cambian, por tanto las rutas que siguen las unidades de la IA pueden llegar a hacerse previsibles. Se plantea una solución que implica

que cada cierto número de turnos se cambiará ligeramente la posición de cada punto guía, recalculando si es posible los valores asociados a dicho punto.

2. Puntos guía inaccesibles: Al colocarse los puntos guía en zonas arbitrarias, es posible que en alguna ocasión un punto guía recaiga en un terreno infranqueable o inaccesible. Esto hace que algunas zonas parezcan imposibles de alcanzar cuando en realidad algunas casillas de esa zona sí podrían ser utilizadas.

3. Rodeos: Algunas casillas que se encuentran en el terreno que rodea a un punto guía pueden estar rodeadas por terreno infranqueable. Acceder a ellas podría requerir dar un rodeo por todo el mapa, a pesar de que por cercanía directa (usando la distancia Manhattan) pertenezcan a un Punto Guía cercano que, en realidad, no posee una cercanía real a este destino.

A pesar de estas desventajas, se considera que, sin ser una solución completa, los Puntos Guía ofrecen una gran cantidad de ventajas que pueden facilitar el trabajo de la IA.

11.3. Toma de decisiones

La toma de decisiones a la hora de elegir órdenes es, sin duda, la principal esencia de este proyecto, y por tanto resulta indispensable explicarla en detalle. Es esta toma de decisiones la que, a la larga, permitirá a la IA ofrecer un reto competitivo para el jugador, y por tanto se ha invertido mayor tiempo y esfuerzo en mejorarla que en otros aspectos del producto final.

Cuando empieza el turno de la IA, esta sigue los siguientes pasos para decidir las acciones que realizar en ese turno.

11.3.1. Carga de unidades

Primero se introducen todas las unidades con las que cuenta el jugador máquina en un vector unidimensional. En este proceso se ignoran las unidades del jugador rival, pues no se tiene un control directo sobre ellas. También se ignoran aquellas unidades que no cuentan con más acciones para ejecutar órdenes, o aquellas que han sido eliminadas.

Solo se tienen en cuenta, por tanto, toda aquella unidad que pueda utilizarse aún y que pertenezca al bando del jugador máquina. Aunque este vector será modificado posteriormente, su distribución inicial es aleatoria.

11.3.2. Elección de un coordinador

El siguiente paso es elegir un coordinador entre las unidades disponibles. Este coordinador se encargará de dirigir a algunas unidades cercanas y establecer órdenes para conseguir que entre todas cooperen y consigan ciertos objetivos comunes. El primer coordinador elegido será aquel que cuente con el mayor valor de prioridad del vector de unidades. En caso de empate, se elegirá arbitrariamente a cualquier unidad que comparta dicho empate.

Todas las unidades del juego tienen asignado un valor de prioridad. Este valor sirve para poder valorar la utilidad, potencia y valor general de la unidad frente a otras. Como ejemplo, los soldados podrían tener asignado un valor de 1, mientras que los tanques, un valor de 4. Una unidad con mayor valor de prioridad que otra será más preciada para la IA y, por tanto, también invertirá más recursos y esfuerzo en mantenerla o en asegurarse de que cuenta con más alternativas que otras unidades cercanas con menor prioridad.

A partir de este momento, se conocerá a la unidad coordinadora como el "jefe".

11.3.3. Elección de subordinados

Una vez elegido el jefe, se debe recorrer el territorio que le rodea para encontrar a sus subordinados. Estas unidades actuarán de un modo servil y actuarán como un solo grupo, dando prioridad siempre al jefe sobre el resto de los miembros del equipo. Asimismo, estas unidades estarán ordenadas en función de su prioridad, de modo que aquellas con valores más altos actúen primero, normalmente.

Para encontrar a las unidades subordinadas se debe atender a dos criterios:

Máximo de unidades

Se establecerá un máximo de unidades para evitar que el coste computacional sea muy alto. De esta manera, cada jefe podría tener solo un N número de subordinados, siendo N un valor arbitrario definido en el diseño de la IA, y fundamentado en el método de ensayo y error hasta dar con uno que permita un coste soportable y que tenga el máximo número de unidades posibles.

Si en la búsqueda de unidades subordinadas se encuentran más de las que el jefe puede controlar, se escogerán solo aquellas que se encuentren más cercanas, hasta llegar a dicho máximo. De esta manera, puede que en un territorio en el que un jefe y su equipo estén actuando se encuentren unidades que no entran dentro de su equipo y que, por tanto, no se tienen en cuenta a la hora de calcular la heurística de las distintas alternativas.

Distancia máxima

Se debe decidir una distancia máxima, a partir de la cuál se dejará de buscar subordinados. Esto se debe a las limitaciones del territorio que domina el jefe, que nunca debe ser mayor de lo que el sistema está dispuesto a soportar.

Por tanto, se podría dar el caso de que un jefe analizara el territorio asignado al completo y encontrara menos unidades subordinadas de las que indica su máximo. Esto es perfectamente viable, y solo indica que el equipo que forman esas unidades tendrá un tamaño menor. Se puede dar el caso de un jefe que solo se dé órdenes a sí mismo, si está lo suficientemente separado del resto de su ejército para no poder ordenarles acciones.

11.3.4. Unidades enemigas

Después de escoger las unidades subordinadas, llega el momento de analizar el terreno de nuevo, esta vez buscando unidades enemigas. Al igual que con las unidades subordinadas, estas nuevas unidades se meten en otro vector para tenerlas controladas y poder utilizarlas en los distintos cálculos de heurística que se realicen. Estas unidades también son ordenadas por orden de prioridad.

Aunque también existen limitaciones a la hora de explorar estas unidades enemigas, estas son significativamente menores que en el caso de las unidades subordinadas.

Máximo de unidades

La presencia de un número alto de unidades enemigas supone un aumento del coste, pero mucho menor al de las unidades subordinadas. Al fin y al cabo, estas unidades no reciben órdenes y, por tanto, solo se tienen en cuenta a la hora de elegir las posibles acciones y al calcular la heurística.

Distancia máxima

La distancia máxima siempre debe ser mayor a la utilizada para las unidades subordinadas, ya que en este caso no solo se debe tener en cuenta las posibilidades en este turno, sino la respuesta del adversario en el suyo. Por tanto, se deben controlar los límites del espacio controlado de tal manera que sean superiores a lo que cualquiera de las unidades del jugador máquina puede recorrer. En caso contrario, podrían cometerse errores fatales como acabar en una posición de peligro amenazada por varias unidades enemigas cercanas que no se han analizado.

11.3.5. Ausencia de unidades enemigas

Si no se han encontrado unidades enemigas, el equipo, como conjunto, deberá buscar a aquella unidad enemiga más cercana (ignorando la niebla de guerra establecida) y acercarse a ella lo máximo posible, como grupo. Este comportamiento no siempre es óptimo, en especial en batallas con objetivos especiales, pero en favor de la sencillez se ha decidido mantenerlo para centrarse en otros aspectos de la toma de decisiones.

11.3.6. Elección de una unidad

El siguiente paso es elegir una unidad subordinada (o el propio jefe) para analizar sus posibilidades. El jefe será el primero en ser elegido, pero luego se irá alternando entre las distintas unidades subordinadas, por orden de prioridad, hasta terminar con aquella que tenga menos prioridad. Cada unidad es elegida en solo una ocasión.

11.3.7. Crear posibilidades

Para la unidad elegida, se crean varios recursos que indican las distintas opciones que tiene dicha unidad.

Array de ataque

En él se indican los enemigos a los que la unidad tiene la posibilidad de atacar. Este array está limitado según el número de unidades que tenga el ejército enemigo (para impedir que su turno sea excesivamente largo) y por el tipo de IA que se esté utilizando. En algunos casos se propondrán más opciones de ataque que en otros. De todas maneras, siempre se intentará, si hay que descartar algunas opciones de ataque por falta de recursos, que las opciones elegidas hagan referencia a los enemigos con mayor prioridad o cercanía.

Array de carga

En esta estructura se hace referencia a las posibles casillas a las que la unidad puede acceder para acercarse más al enemigo. Al igual que el array de ataque, se ve condicionado por el número de unidades y el tipo de IA, por lo que su tamaño podrá variar según cambien estos valores. Si hay que descartar movimientos, se empezará a descartar siempre aquellos que supongan un avance menor respecto a la posición de la unidad.

Acciones de protección

Por último, en esta estructura se almacenarán distintas opciones de protección que la unidad puede realizar para moverse cerca de su jefe y escudarle contra ataques enemigos directos. En ellas,

la unidad siempre intentará colocarse junto a su jefe, o lo más cerca posible. Al igual que en los otros casos, el número de acciones generadas depende del tamaño del ejército y del tipo de IA.

11.3.8. Heurística inicial

Tras definir las distintas opciones que tiene la unidad, se realiza un breve chequeo de la zona para extraer la heurística inicial, antes de ejecutar ninguna acción. Cuanto mayor sea la heurística, menos deseable será dicho estado de juego. Para medirla, solo se tienen en cuenta las unidades subordinadas y los enemigos analizados. Si hay más unidades en el terreno analizado, se ignoran a la hora de calcular dicha heurística (aunque no a la hora de bloquear posibles casillas, por ejemplo. La heurística tiene en cuenta los siguientes factores:

Unidades amenazadas

Por cada unidad subordinada (incluyendo al jefe) que esté siendo amenazada por uno de los enemigos, la heurística aumentará en un valor que depende directamente de la prioridad de la unidad y el daño medio que recibiría en dicho ataque, multiplicando luego dicho valor por otro que depende exclusivamente del tipo de IA que se use. En una primera iteración se realizaba una comprobación intensiva para comprobar qué unidades estaban amenazadas siguiendo las reglas de movimiento, pero el alto coste de recursos al aplicar en muchas ocasiones la heurística suponía que el proceso fuera lento y costoso. Por eso, en este ámbito se considera que todos los terrenos tienen un coste de movimiento de 1.

Esta solución dista mucho de ser óptima. Para empezar, impide comprobar si una unidad está cubriendo a otra. Además, no tiene en cuenta las características del terreno, por lo que podría llegar a considerar como peligrosa una casilla segura. Por ello, se ha implementado una versión más pesada de la heurística que solo se aplica en casos concretos y aislados.

Cercanía a enemigos

La idea original es la siguiente: La heurística también se incrementaría dependiendo de lo cerca que estuviera cada unidad aliada de la unidad enemiga con mayor prioridad. Para calcular este valor se mediría la distancia real (teniendo en cuenta el coste del terreno) entre la unidad y cualquiera de las casillas desde las cuales se pudiera atacar al enemigo con mayor prioridad, valor que luego se multiplicaría por otro valor arbitrario que dependería del tipo de IA. De esta manera, cuantas más acciones debiera invertir dicha unidad para atacar al enemigo más prioritario, más subiría la heurística. Si, por algún motivo, no se pudiera atacar al enemigo con mayor prioridad, se analizaría la cercanía con el segundo, y así. Si ningún enemigo pudiera ser alcanzado de ninguna manera, o si se superara un número de comprobaciones establecido en cada tipo de IA, no se sumaría ninguna cantidad a la heurística.

Sin embargo, tras varias iteraciones se observó que este proceso imponía mucha carga en el cálculo de la heurística, de modo que se simplificó para facilitar el proceso.

11.3.9. Elegir una opción

Ahora toca decidir qué orden elegir entre las que se han escogido previamente. Este conjunto de acciones estará, en la mayoría de los casos, limitada a una serie de opciones prometedoras, habiendo descartado previamente otras para hacer del cálculo algo viable.

Para cada opción se calcula una nueva heurística, aplicando ciertas consecuencias que se corresponden a lo que se prevé que ocurrirá al realizar la acción. Esta nueva heurística está influenciada

por el daño que la unidad ha causado al enemigo, modificado por la prioridad de dicho enemigo y el tipo de IA, si la opción elegida es un ataque, o por el número de casillas recorridas, modificado también por el tipo de IA, si la opción elegida es un movimiento. De esta manera, se puede usar este valor hipotético como comparativa con la heurística inicial y con la de otras opciones.

Se elige la opción con mejor heurística. Esto puede dar lugar a distintas alternativas, que dependen de varios factores, entre ellos la importancia que del tipo de IA a cada acción. Algunas de las órdenes más comunes serían atacar a enemigos con alta valía, escoltar a unidades aliadas con mayor valía, moverse a una zona no amenazada con buena cobertura que acerque a la unidad a un posible enemigo, hacer guardia para mejorar la cobertura de la unidad en el siguiente turno, atacar indiscriminadamente a la unidad más cercana, etcétera.

En caso de que ninguna heurística mejore la inicial, se puede establecer una opción por defecto, que en caso de los tipos de IA más agresivos podría consistir en un ataque indiscriminado, mientras que en otras versiones más prudentes podría implicar una guardia.

11.3.10. Dar la orden

Una vez escogida la acción, se da la orden a la unidad para que la cumpla. En algunos casos, será la propia unidad la que decida los detalles de la orden, como el camino que escoge para cumplirla o el terreno más favorable, pero en otros la orden será directa y la unidad la ejecutará con exactitud. Las consecuencias de dicha orden deben de aplicarse inmediatamente, por ejemplo retirando unidades eliminadas para que no afecten de nuevo en las comprobaciones futuras.

En situaciones concretas se podría decidir que la acción elegida fuera esperar, de tal manera que la unidad no ejecuta ninguna orden y no gasta acciones, pero pasa el turno para que otras puedan actuar. Esta opción permitiría que dicha unidad formara parte de otro equipo posterior en el que sus expectativas sean mayores, pero debe de usarse con cuidado: si muchas unidades se quedan esperando, el coste temporal probablemente aumentaría de forma significativa.

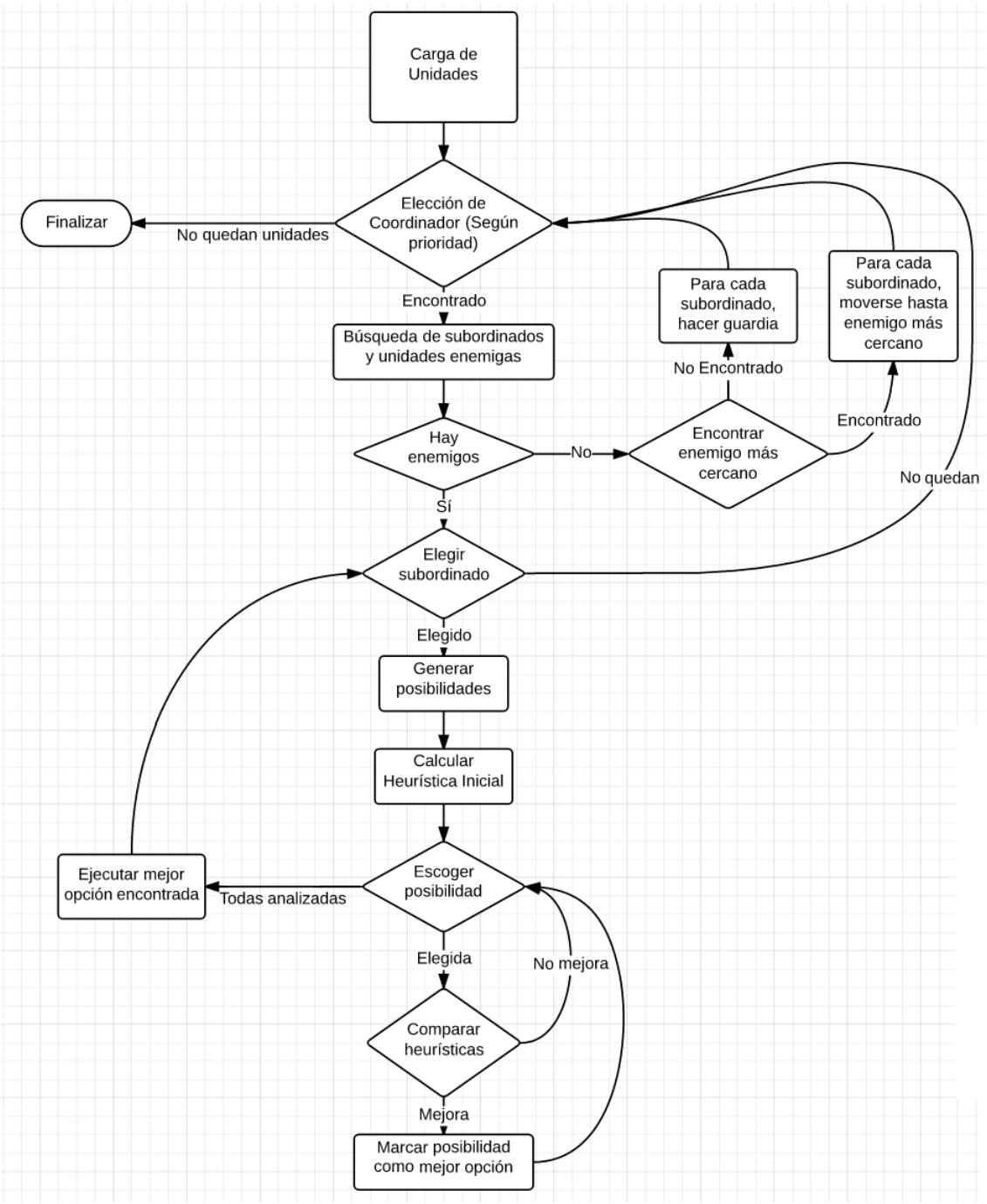
11.3.11. Repetir

En caso de que aún queden unidades en el equipo que no han actuado todavía, se vuelve al punto de elegir una unidad, hasta que todas hayan realizado una acción como mínimo (o hayan decidido esperar). Una vez se haya terminado, se pasa al siguiente punto.

11.3.12. Elegir un nuevo coordinador

Las unidades que ya no tienen acciones restantes son sacadas del array general de unidades, y se vuelve al paso de elegir un jefe. Es posible que las unidades con mayor prioridad sean jefe en varias ocasiones en un mismo turno, pero es algo que se espera: Al fin y al cabo, son las unidades más valiosas del ejército.

Una vez el array de unidades esté vacío, todas las unidades habrán gastado sus acciones y, por tanto, se dará el turno por terminado.



11.3.13. Dificultades de implementación

Estos son algunos de los escollos con los que se encuentra este método de toma de decisiones.

Límite de tiempo

El algoritmo puede llegar a ser costoso cuando hay una gran cantidad de unidades o acciones posibles. Se plantean dos posibilidades para poder paliar esto, que ya han sido integradas en el sistema de forma más o menos similar a lo aquí especificado.

La primera consiste en reducir el número de "pasadas". Cada vez que se elige un nuevo jefe se considera que se ha realizado una nueva pasada, y se permite que dicho jefe y su equipo ejecuten sus acciones de forma habitual. Sin embargo, al llegar a cierto número de pasadas, se entiende que las unidades con mayor prioridad del ejército (y aquellas que han sido sus subordinadas) ya han actuado, por lo cual las unidades con acciones restantes tienen una prioridad relativamente baja. Si se quiere ahorrar tiempo, se podría sacrificar eficiencia al ordenar a estas unidades una acción por defecto, como atacar indiscriminadamente o asegurar su posición, para dar por finalizado el turno. Otra alternativa para implementar esta solución (que se a declinado a favor de la anterior) sería especialmente efectiva si se cuenta con un control del tiempo total invertido, y podría aplicarse al llegar a cierto hito temporal en vez de contar las pasadas.

La segunda propone una solución a nivel más individual, en el que cada unidad evaluaría, después de ejecutar sus órdenes, si sus acciones restantes superan una cantidad arbitraria (por ejemplo, un cuarto de sus acciones totales). De no ser así, podría aplicar una o más acciones por defecto y gastar el resto de sus acciones para no volver a ser elegida como jefe o subordinado, dando paso así a aquellas unidades que todavía tienen un porcentaje considerable de acciones restantes. Esta acción por defecto podría ser atacar a una unidad en su rango si tiene suficientes acciones y considera que el contraataque no va a ser alto, o hacer una guardia. Estas acciones por defecto podrían depender del tipo de IA.

Límite de memoria

En algunos casos, la cantidad de funciones recursivas y el gasto de espacio hace que, al comprobar las posibilidades de varias unidades, el gasto de memoria haga que el videojuego vaya más lento que de costumbre. Para ello se deben aplicar controles similares a los propuestos en el punto anterior, evitando así que el juego se quede bloqueado o reduzca su eficiencia de forma drástica.

11.4. Ejecución de órdenes

A la hora de ejecutar una orden, el algoritmo da control a la unidad correspondiente, que a partir de ese momento es la encargada de cumplir la orden y efectuar los cambios en el estado de juego según los resultados de esta. Cuando ha terminado de ejecutarla, es ella misma la que devuelve el control del juego al algoritmo.

En ciertas ocasiones, pueden darse varias órdenes seguidas para una o varias unidades. Estas órdenes deberán tener la consideración de no pisarse entre ellas, de manera que no den lugar a combinaciones no esperadas o errores en el juego. En estos raros casos en los que una unidad termina de ejecutar una acción y hay otras en la çola de espera", en vez de devolver el control al algoritmo, la unidad llama a la siguiente en dicha cola para que efectúe su acción, siendo la unidad que realiza la última la que se encarga de devolver el control al algoritmo.

11.5. Distintos tipos de IA

Como ya se ha mencionado en varias ocasiones en este mismo capítulo, se requieren una serie de valores arbitrarios que sirvan como orientación para el algoritmo a la hora de calcular la heurística,

elegir y descartar opciones y controlar recursos como el tiempo y la memoria. Estas constantes podrían definirse en cada parte del algoritmo, pero se ha optado por incluirlas en una serie de funciones cuyo valor devuelto depende de un nuevo factor: el tipo de IA.

De esta manera, se pueden crear distintos perfiles para comprobar su efectividad. Todos usan los mismos procesos, pero varían en cuanto a su valoración de riesgos y beneficios. Un tipo de IA agresiva no aumentaría tanto la heurística al tener a sus unidades amenazadas, mientras que otro más prudente probablemente pondría un valor reducido al daño causado en los ataques.

Se han definido, a estas alturas, cinco tipos de IA distintas, para abarcar cinco estados que van desde una valoración prudente y un alto interés por la conservación de las unidades en un extremo, hasta una IA marcadamente ofensiva que ofrece un peso especialmente alto a las acciones de ataque en la heurística.

11.6. Pruebas realizadas

Se han realizado pruebas en cinco niveles distintos. La IA responde bien en la mayoría de ellos, aunque su efectividad disminuye en algunos casos. Estas son algunas de las conclusiones.

11.6.1. Unidades apiñadas

La IA muestra ciertos problemas a la hora de tratar con grupos grandes de unidades situadas en espacios reducidos. Al buscar siempre las opciones de ataque hacia aquellas unidades con mayor prioridad, suele obviar a otras con prioridades más bajas. Esto hace que, en algunos casos en los que dichas unidades prioritarias son inaccesibles o se encuentran en territorios peligrosos (amenazados por muchas unidades, y por tanto con una alta heurística), la unidad decida que atacar no es útil. Sin embargo, a veces se da el caso de que se pudiera atacar a una unidad con baja prioridad cercana de forma efectiva y sin ponerse en riesgo. Sin embargo, normalmente estas opciones se han descartado por las limitaciones antes mencionadas, y muchas veces no son tenidas en cuenta en estos casos.

11.6.2. Efectividad

La efectividad de la IA varía. Aquellos tipos más prudentes han demostrado una menor competencia en lo referente al resultado final, pues suelen mantenerse alejadas de las unidades enemigas a menos que tengan una oportunidad segura de causar daños sin arriesgarse. Esta falta de riesgo hace que el rival humano sea capaz de controlar zonas críticas con facilidad. Los tipos más agresivos han obtenido mejores resultados, pues suelen tomar más riesgos, pero sin dejar de buscar buenas posiciones cuando los ataques no son posibles.

Capítulo 12

Conclusiones

12.1. Un producto acabado

Una vez se cuenta con el videojuego completo, con la IA integrada, y se han realizado suficientes pruebas, es fácil extraer una serie de conclusiones aquí listadas. Para poder analizarlo de forma ordenada se evaluarán los requisitos mencionados en el capítulo correspondiente, estableciendo un valor numérico entre el 0 y el 10 para cada uno, representando el 0 un fracaso absoluto a la hora de realizar dicho requisito y el 10 un éxito rotundo.

12.2. Requisitos generales

12.2.1. Gráficos

Variedad

Las unidades y terrenos quedan bien diferenciados en el mapa, y la variedad de elementos es notable. Como punto a mejorar, algunos menús resultan monótonos, y las animaciones de introducción en cada batalla se vuelven repetitivas. Se podrían buscar otras alternativas para aumentar la diversidad.

Valoración personal: 7.

Complejidad

La complejidad de los gráficos es adecuada, si bien la poca resolución de estos (la mayoría son sprites de 32 bits) limita la calidad que pueden llegar a tener. Se podría incrementar el tamaño de los mapas, mostrándolos luego con zoom reducido usando el sistema de vistas de Game Maker Studio, de modo que se mejore la calidad de dichos gráficos sin necesidad de cambiar el estilo de juego o la apariencia general de este.

Valoración personal: 8.

12.2.2. Jugabilidad

Variedad

El número de tipos distintos de unidades y terrenos supera lo propuesto, y la variedad de estas es aceptable. Como punto a mejorar, se podría aumentar la variedad de órdenes que se pueden dar a una unidad, introduciendo por ejemplo una función para algunas unidades especiales (como el vehículo médico) que permita curar a otras, o una orden específica que se pueda realizar en algunos tipos de terreno, como colocar alambradas en un pueblo, que permita reducir el tipo de movimiento de unidades enemigas en dicho terreno o mejorar de forma permanente la cobertura de este.

Valoración personal: 7.

Duración

La duración del juego se corresponde a lo requerido, contando con una variedad de niveles adecuada para probar las capacidades de la IA en distintos terrenos. Se podrían introducir nuevos niveles para hacer la progresión de dificultad más gradual y para probar nuevos tipos de mapas, como mapas no cuadrados o con elementos geográficos relevantes.

Valoración personal: 9.

Reglas

Las reglas están bien definidas y el tutorial es claro. Sin embargo, este resulta algo monótono y nada progresivo, dificultando el aprendizaje al tener que .estudiar"dichas reglas antes siquiera de poder empezar a jugar. Introducir un tutorial interactivo mejoraría la experiencia de juego.

Valoración personal: 7.

12.2.3. Plataforma

Instalación

La instalación es sencilla e intuitiva, y se puede realizar en pocos pasos. El instalador realiza el proceso de forma automática. Se podrían proponer alternativas para instalar el juego en otras plataformas, pero habría que mirar las limitaciones de Game Maker.

Valoración personal: 10.

Requisitos técnicos

Aunque el juego ha sido evaluado en distintos ordenadores, los requisitos técnicos no han sido analizados en profundidad, por lo que su comportamiento podría resultar inesperado en según qué soporte.

Valoración personal: 4.

12.2.4. Inteligencia artificial

Herramientas teóricas

Se han evaluado varios algoritmos, de los cuales al final se han elegido tres distintos (por lo que el requisito se considera cumplido). Sin embargo, la complejidad de dichos algoritmos es menor que la de otros más avanzados desarrollados posteriormente, y el trabajo de investigación podría haberse aumentado para considerar otras posibles opciones.

Valoración personal: 6.

Entorno desconocido

Todos los indicadores propuestos para este requisito han sido cumplidos y sus propuestas introducidas en el juego. El entorno al que se enfrenta la IA es lo suficientemente caótico como para que esta deba de adaptarse a situaciones imprevistas. Sin embargo, siempre se puede añadir una mayor variedad de unidades y ampliar ciertas reglas para que permitan nuevas alternativas inesperadas, como ataques fallidos o moral en las unidades.

Valoración personal: 9.

Tiempo de espera

Los tiempos esperados se han cumplido en las distintas pruebas realizadas, a excepción del tiempo de carga, que en mapas muy grandes puede llegar a ser algo lento debido a la necesidad de implementar los Puntos Guía. Este tiempo, a pesar de todo, es mejorable, y con algunas revisiones se podrían optimizar algunos de los procesos de la IA para reducirlo aún más sin sacrificar demasiada eficiencia.

Valoración personal: 6.

Tipos de IA

Aunque la IA dinámica queda por implementar, se han podido escoger varios tipos de IA que ofrecen una diversidad de oponentes adecuada. Sin embargo, siempre se podrían crear más perfiles que no solo se basen en el factor Riesgo/Agresividad", sino que tengan otros aspectos en cuenta, como la consecución de objetivos. Al fin y al cabo, la estrategia es un campo complejo con muchos factores, y de esta manera queda algo simplificada. También se podría intentar que el tipo de IA influya más en varios aspectos del algoritmo de toma de decisiones, en vez de únicamente en las directrices que se utilizan para evaluar dichas opciones. Por ejemplo, algunos tipos de IA complejos podrían añadir un nuevo tipo de orden compuesta.

Valoración personal: 5.

12.2.5. Otros requisitos

Facilidad de análisis

Las características esenciales de la IA se pueden evaluar fácilmente al jugar un nivel contra ella. Sin embargo, es cierto que en algunos niveles los aspectos más complejos de dicha inteligencia pueden dar lugar a comportamientos a veces inesperados, que entorpecen la evaluación al no tener un contexto claro. Este tipo de comportamientos inesperados debería evitarse.

Valoración personal: 7.

12.2.6. Requisitos subjetivos

Gráficos atractivos

Los gráficos son coloridos y ofrecen un soporte agradable. La falta de animaciones y las similitudes con otros juegos del género, sin embargo, son defectos a tener en cuenta a la hora de mejorarlos.

Valoración personal: 6.

Gráficos identificables

En todo momento queda claro lo que es cada unidad en el juego, y con un simple vistazo se pueden divisar los elementos esenciales. Como pega, los distintos valores numéricos de cada unidad (acciones, vida...) hacen que a veces el conjunto quede algo recargado y confuso. El hecho de que algunos iconos sean imágenes de alta resolución reducidas hace que estos iconos sean menos identificables que el resto, sin por ello llegar a ser confusos.

Valoración personal: 8.

Entorno sencillo

Aunque con un poco de práctica es fácil adaptarse al entorno, en ocasiones este resulta poco intuitivo. Las distintas ayudas visuales que se ofrecen, como la iluminación de casillas, son a veces algo caóticas, y la interfaz a veces puede resultar ligeramente lenta, pudiendo implementarse cambios para permitir a jugadores experimentados realizar sus turnos de manera más rápida y eficiente. Por último, es necesario plantearse la posibilidad de mostrar el rango de acción de cada unidad enemiga cuando el usuario así lo requiera, para evitar tener que calcularlo manualmente.

Valoración personal: 5.

Desafío

El grado de desafío varía dependiendo del nivel, como se había establecido. Sin embargo, la curva de dificultad es alta, de manera que en el tercer nivel se pasa de luchar en superioridad de condiciones a enfrentarse a un ejército significativamente más poderoso que el del jugador con un número limitado de unidades. Este salto de dificultad puede frustrar a muchos jugadores, por lo que sería recomendable introducir otros niveles entre medias para que esta subida de dificultad sea más progresiva, o reducir la dificultad del tercer nivel.

Valoración personal: 6.

Mecánicas sencillas

Las mecánicas son simples y fáciles de entender una vez dominadas. El sistema de acciones puede resultar algo complejo en un primer momento, y podría simplificarse.

Valoración personal: 8.

Entretenimiento

El juego resulta entretenido, aunque se podrían añadir más elementos lúdicos como una ambientación mayor, eventos especiales o una mayor variedad de música.

Valoración personal: 7.

Anticipación de la IA

La IA es capaz de adaptarse a algunos planes sencillos, pero en otras ocasiones la heurística concreta de ese tipo de IA hace que tome decisiones inesperadas. Deberían realizarse más pruebas para refinar el peso que se da a cada elemento de la heurística y añadir nuevos parámetros que tener en cuenta.

Valoración personal: 6.

Potencia de la IA

Aunque la IA es capaz de ofrecer un reto aceptable, en igualdad de condiciones sigue siendo claramente inferior a un humano. Puede que jugadores novatos encontrarán difícil alcanzar la victoria en una situación así, pero para el jugador experimentado resulta relativamente simple si no se incrementa el número de efectivos del enemigo.

Valoración personal: 5.

12.3. Valoración final

Aunque la satisfacción con el resultado final es alta, la cantidad de modificaciones que se pueden idear para mejorar la potencia del sistema de toma de decisiones hacen pensar que, con el tiempo debido, se podría crear una inteligencia artificial mucho más eficiente, que pudiera llegar a mejorar la experiencia de juego.

Sin embargo, se considera que se ha logrado implementar un boceto muy coherente con la idea esencial del proyecto, y lo aprendido en el proceso hace que la impresión general sea bastante positiva.

Capítulo 13

Posibles Mejoras

13.1. Propuesta de mejora

Algunas de las ideas que han surgido a lo largo del resto de capítulos, y durante la realización del proyecto, se presentan aquí como posibles líneas mediante las cuales continuar mejorando el videojuego y, especialmente, la capacidad del sistema de toma de decisiones elegido. Estas propuestas quedan .[en] el tinteroçomo futuras implementaciones, que a pesar de no ser las únicas que el lector pudiera llegar a listar, sí ofrecen una perspectiva de mejora basada en aspectos prometedores.

13.2. Mejora de la IA

13.2.1. Creación de batallones

Una de las posibles mejoras que podrían introducirse en la IA es la búsqueda de batallones "semi-fijos", que sean un reflejo de los equipos liderados por los distintos jefes pero tengan una naturaleza menos mutable y más duradera.

La justificación de esta idea reside en la filosofía de que en la variedad está la fuerza. Así, cada turno podrían dedicarse una breve cantidad de recursos a intentar agrupar un grupo de unidades distintas (con distinta prioridad) en un batallón. Este batallón no sería algo fijo, y podría deshacerse o modificarse en turnos posteriores, pero sí ofrecería un punto de referencia a la hora de mover esas unidades, que intentarían mantenerse juntas en todo momento y actuar bajo las órdenes del mismo jefe siempre que puedan.

De esta manera, se combate la naturaleza intermitente de los equipos temporales que forman los coordinadores, creando asociaciones de unidades que cooperan durante más de un turno con el objetivo de obtener beneficios a largo plazo.

13.2.2. IA dinámica

Como se ha insinuado en capítulos anteriores, es probable que la IA mejorase de poder cambiar su "personalidad.[a] mitad del juego, según el estado de este. De esta manera, el tipo de IA variaría de forma dinámica, volviéndose más agresiva o prudente según el comportamiento del adversario o la diferencia de fuerzas existente entre ambos bandos.

13.2.3. Objetivos avanzados

La IA actual considera como objetivos principales la conservación de sus propias unidades y la eliminación de las que pertenecen al bando enemigo. Sin embargo, en ocasiones estos objetivos no son tan prioritarios. Por ejemplo, sacrificar una unidad podría ser un objetivo válido siempre y cuando ello supusiera una ventaja táctica, como controlar un terreno prioritario o llevar al enemigo a una trampa. La introducción de objetivos avanzados podría influir en la heurística, o actuar como un algoritmo independiente que estuviera estrechamente relacionado con el principal, de tal modo que la IA empezara a mostrar tácticas avanzadas frente a su adversario.

13.2.4. Comportamientos predefinidos

Aunque la idea del proyecto era hacer un sistema de toma de decisiones general y alejado de las técnicas comúnmente conocidas como "scripting", es cierto que la implementación de pequeñas "trampas"de juego, como una serie de plantillas que puedan adaptarse a las distintas situaciones, mejoraría la efectividad de la IA en más de una situación. Comportamientos individuales como "golpear y alejarse.º conjuntos como ünа unidad ataca, la otra le protege"pueden ser definidos sin mucho esfuerzo y dar lugar a resultados interesantes.

Por otro lado, abusar de esta técnica podría llegar a ser contraproducente. Es habitual que los jugadores aprendan de sus errores y acaben anticipándose a estos rígidos comportamientos, lo que disminuiría de forma drástica la efectividad de la IA contra ellos y dando lugar a que esta pueda ser engañada por rivales astutos que exploten su naturaleza determinística. Para evitarlo, se podrían usar solo en condiciones muy concretas, o dejar dicha decisión al azar. En cualquier caso, es una propuesta de mejora que debe implantarse con cuidado.

13.2.5. Tratamiento específico de unidades

Aunque va en contra de la filosofía subyacente en este proyecto, la posibilidad de realizar un tratamiento diferenciado para cada tipo de unidad aumentaría la efectividad del jugador máquina y, posiblemente, también el valor lúdico del videojuego. De esta manera, se podría considerar que el entorno conocido no va a cambiar demasiado, y adaptar un sistema de valoraciones específicas para cada unidad en dicho entorno. Un ejemplo sería "las unidades de artillería necesitan más protección que los tanques.º "las fuerzas especiales siempre deben de quedar protegidas de los ataques enemigos al acabar el turno".

13.2.6. Ordenes complejas

Contemplar la posibilidad de introducir en el sistema un nuevo conjunto de órdenes compuestas para la IA presenta varios beneficios obvios. El primero, la facilidad de modificación del algoritmo, cuya estructura quedaría mucho más clara al tener menos líneas de código. El segundo, la posibilidad de una unidad de ejecutar una serie de comportamientos predefinidos cuya eficacia esté demostrada, como atacar en varias ocasiones si se prevé un daño alto.

13.2.7. Vida restante

A la hora de medir la prioridad de una unidad, su vida debería aparecer como un posible factor decisivo. Por muy prioritaria que sea, al fin y al cabo, una unidad resulta mucho menos útil cuando su vida ha descendido considerablemente. No solo baja su efectividad en combate, sino que resulta

más fácil de eliminar por ataques enemigos y, por tanto, no ofrece muchas garantías a la hora de aguantar la posición o proteger un terreno o a una unidad aliada.

13.2.8. Array de ataques cercanos

Para evitar el problema que se menciona en la sección de evaluación de la IA, relacionado con las unidades agrupadas en zonas muy reducidas, se podría añadir un array de opciones adicional. Este array sería similar al array de ataques, y también tendría un límite de opciones que vendría dado por el tipo de IA. Sin embargo, y a diferencia de su homólogo, a la hora de descartar opciones siempre empezaría por aquellas que hicieran referencia a unidades más lejanas, empezando siempre por las cercanas. De esta manera, se podrían evitar situaciones extrañas en las que una unidad no ataca a otra que tiene al lado porque dicho enemigo tiene una prioridad baja, y por tanto la opción de atacarle ni siquiera se ha tenido en cuenta.

13.3. Otras mejoras

13.3.1. Aspecto del juego

Es fácil idear una serie de correcciones o añadidos para mejorar el aspecto general del juego, como cambiar las imágenes e iconos de las unidades por otros más atractivos u originales, y mejorar la calidad de los menús. De esta manera, el juego ofrecería un aspecto más profesional y atraería a más jugadores.

13.3.2. Nuevos niveles

Aunque para el proyecto no se tenían planeados muchos más, para un juego completo la cantidad de niveles ofertada es relativamente pequeña. Sería interesante añadir nuevos niveles, así como una campaña ambientada con una historia interesante.

13.3.3. Modo multijugador

Un nuevo modo para permitir a dos o más jugadores humanos participar en una batalla no sería difícil de implementar, y mejoraría la experiencia de juego haciendo que la vida útil de este sea mayor antes de que el jugador se aburra.

13.3.4. Editor de niveles

Crear una herramienta de edición de niveles permitiría a los usuarios el poder dar rienda suelta a su creatividad, y establecer las condiciones bajo las cuales desean jugar una batalla. Aunque esta herramienta requeriría cierta dedicación, los beneficios son también altos, pues permitirían alargar el juego todo lo que se desease al general cada usuario nuevos desafíos que superar.

13.3.5. Nuevos modos de juego

Se podrían diseñar nuevos tipos de partida en los que el objetivo no únicamente la aniquilación del enemigo, sino que se contara con ciertos requisitos adicionales. Batallas de capturar la bandera, control de puntos estratégicos o cumplir una serie de condiciones en un número de turnos limitados son solo ejemplos de lo que podría idearse para ampliar la experiencia de juego. Esta modificación

requeriría a su vez una adaptación de la IA para que se adapte a estos nuevos modos, a menos que estos sean especialmente orientados al combate entre jugadores humanos.

13.3.6. Enfoque profesional

Es fácil pensar en una serie de opciones que harían que el aspecto del juego fuera más comercial y, por tanto, susceptible a ser introducido en el mercado. Esta propuesta de mejora es en realidad un conjunto de ideas que podrían aplicarse, como mejorar el aspecto del menú principal, adaptar el juego a otras plataformas o sistemas operativos, utilizar un paquete de gráficos especialmente diseñado para el juego, mejorar el repertorio musical, etcétera.

Capítulo 14

Presupuestos

14.1. Implementación del juego

A continuación se ofrece un proyecto de implementación que ofrece, de forma aproximada, una idea general de los costes a la hora de aplicar las distintas mejoras al juego y dejarlo listo para ser comercializado. En él no se especifican las mejoras a aplicar, pues se entiende que dicha lista deberá ser generada tras el proceso de análisis, aunque siempre utilizando como referencia el capítulo de mejoras en esta misma memoria.

El proceso de implementación de mejoras durará el equivalente a dos meses de trabajo (ocho semanas) y estará dividido en cuatro fases establecidas posteriormente. No se realizará un proyecto de comercialización, limitando el presupuesto a la producción del software.

14.2. Presupuesto

En este presupuesto se han tenido en cuenta distintos tipos de coste:

- Salarios
- Coste de Seguridad Social
- Recursos Materiales
- Imprevistos y Costes Indirectos

14.2.1. Salarios

Los salarios varían según el rol de cada empleado. Se estima que los costes totales, en lo que respecta al salario bruto y la seguridad social de los empleados, en cada fase son:

- Análisis: 1924.95€ (1157.41€ del Jefe de Proyecto y 767.54€ del Analista).
- Diseño: 1820.78€ (767.54€ del Analista, 526.62€
  EURtm del Diseñador y 526.62€ del Diseñador Gráfico).
- Implementación: 4962.45€ (767.54€ del Analista, 526.62€ del Diseñador, 1561.8€ del Programador y 2106.48€ del Diseñador Gráfico).

- 2222.06€ (462.96€ del Jefe de Proyecto, 767.54€ del Analista, 780.9€ del Programador y 210.65€ del Diseñador gráfico.

Coste total de salarios (para la empresa): 10930,24€.

14.2.2. Coste de Seguridad Social

El coste de la Seguridad Social equivale a un 30 % de los salarios, aproximadamente: 3279,07€.

14.2.3. Recursos Materiales

- Equipo Informático: 2400€
- Coste licencias: 150€
- Material fungible: 40€
- Impresiones: 40€

Coste total : 2630€
par

14.2.4. Imprevistos y Costes Indirectos

Los costes indirectos e imprevistos se estiman en un 10 % del coste total de salarios y recursos materiales (13560.24€). Algunas de las áreas que pueden abarcar estos costes son posibles despidos y finiquitos, cualquier impuesto de sociedades aplicable, etcétera.
Coste total : 1356.02€

14.2.5. Coste total del proyecto

El coste total del proyecto se estima en 18195,33€ euros.

14.3. Cálculo de Salario

Aquí se explica de forma abreviada el método que se ha seguido para calcular el coste de salarios para la empresa. Las características del trabajo realizado serán las siguientes:

- Horas trabajadas por día: 8 horas.
- Fines de semana: (2 días/semana * 52 semanas/año) : 104 días/año.
- Días festivos: 14 días (durante todo el año).
- Vacaciones: 31 días (Mes de Agosto).
- Días laborables: (365 -104 - 14 - 31) : 216 días.
- Horas laborables: 216 días * 8 horas/día : 1728 horas.

El salario bruto se calculará teniendo en cuenta los siguientes factores:

- IRPF: Varía según el salario bruto.
- Seguridad Social: 5 % sobre el salario bruto.
- Pagas: 14.

La fórmula para hallar el salario bruto anual sería:

$$SN : (SBA - SBA * IRPF - SBA * 0{,}05)/14$$

Despejando queda que el Salario Bruto se calcula con la siguiente expresión:

$$SBA = (SN * 14)/(0{,}95 - IRPF))$$

A la hora de calcular el coste por hora, se debe dividir esta última cifra entre las horas laborables (1728).

14.3.1. Roles

Los distintos roles que se plantean son:

Jefe de Proyecto

- Salario neto: 2500€
- IRPF: 25
- Coste por hora: 28.93€

Analista

- Salario neto: 1800€
- IRPF: 19
- Coste por hora: 19.19€

Diseñador

- Salario neto: 1300€
- IRPF: 15
- Coste por hora: 13.16€

Programador

- Salario neto: 1000€
- IRPF: 12
- Coste por hora: 9.76€

Diseñador Gráfico

- Salario neto: 1300€
- IRPF: 15
- Coste por hora: 13.16€

14.3.2. Fases del proyecto

Las distintas fases, con el coste de cada una incluido, son:

Análisis

Objetivos: Realizar el análisis del producto así como de las mejoras a implementar.

- Jefe de proyecto: 40 horas. Coste : 1157.41€
- Analista: 40 horas. Coste : 767.54€

Coste total : 1924.95€

Diseño

Objetivos: Realizar el diseño de la implementación para cada una de las mejoras.

- Analista: 40 horas. Coste : 767.54€
- Diseñador: 40 horas. Coste : 526.62€
- Diseñador gráfico: 40 horas. Coste : 526.62€

Coste total : 1820.78€

Implementación

Objetivos: Implementar el diseño propuesto en la fase anterior.

- Analista: 40 horas. Coste : 767.54€
- Diseñador: 40 horas. Coste : 526.62€
- Programador: 160 horas. Coste : 1561.8€
- Diseñador gráfico: 160 horas. Coste : 2106.48€

Coste total : 4962.45€

Pruebas

Objetivos: Fase de pruebas y corrección de errores.

- Jefe de proyecto: 16 horas. Coste : 462.96€
- Analista: 40 horas. Coste : 767.54€
- Programador: 80 horas. Coste : 780.9€
- Diseñador gráfico: 16 horas. Coste : 210.65€

Coste total : 2222.06€

Bibliografía

[1] Ignacio Sánchez-Cuenca. *Teoría de juegos*. Cuadernos Metodológicos. Centro de Investigaciones Sociológicas, 2009.

[2] Ernest Davis. «The Singularity and the State of the Art in Artificial Intelligence». En: (2013). URL: https://www.cs.nyu.edu/davise/papers/singularity.pdf.

[3] Jon Brodkin. *How Unity3D Became a Game-Development Beast*. URL: http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/ (visitado 21-05-2015).

[4] Dani Candil. *Cuatro motores gráficos para perder el miedo y lanzarse al desarrollo de videojuegos*. URL: http://www.vidaextra.com/listas/4-motores-graficos-para-perder-el-miedo-y-lanzarse-al-desarrollo-de-videojuegos (visitado 10-06-2015).

[5] Ernesto Cotos. *Algoritmos básicos de grafos*. URL: http://ccg.ciens.ucv.ve/~ernesto/nds/CotoND200302.pdf (visitado 11-09-2015).

[6] Gur Dotan. *Top 10 Unity Games Ever Made*. URL: http://blog.soom.la/2015/01/top-10-unity-games-ever-made.html (visitado 21-05-2015).

[7] Justin Eldridge. *Strategy game genre, definition and meaning*. URL: http://www.examiner.com/article/the-definition-and-meaning-of-the-strategy-game-genre (visitado 02-03-2015).

[8] Jason Fox. *Tic Tac Toe: Understanding The Minimax Algorithm*. URL: http://neverstopbuilding.com/minimax (visitado 11-09-2015).

[9] Indie Games. *CryEngine 3 vs Unreal Engine 4*. URL: http://indiegames.com/2013/02/its_official_xna_is_dead.html (visitado 28-05-2015).

[10] Ashley Gullen. *The future of Construct*. URL: https://www.scirra.com/blog/155/the-future-of-construct (visitado 06-09-2015).

[11] Donald Kehoe. *Designing Artificial Intelligence for Games (Part 1)*. URL: https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1 (visitado 11-09-2015).

[12] Universidad de Málaga. *Introducción a la Teoría de Juegos*. URL: http://www.eumed.net/cursecon/juegos/ (visitado 11-09-2015).

[13] Tom Marchin. *Construct 2 Review 2014/2015 – Is it any good?* URL: http://www.burquitlambadgers.com/2014/10/construct-2-review/ (visitado 06-09-2015).

[14] Mark Masters. *Unity, Source 2, Unreal Engine 4, or CryENGINE - Which Game Engine Should I Choose?* URL: http://blog.digitaltutors.com/unity-udk-cryengine-game-engine-choose/ (visitado 25-05-2015).

[15] Mark Masters. *Unreal Engine 4 - Powering The Next Generation of Games.* URL: http://blog.digitaltutors.com/unreal-engine-4-powering-next-generation-games/ (visitado 25-05-2015).

[16] Bo Moore. *No coding required: How new designers are using GameMaker to create indie smash hits.* URL: http://www.pcgamer.com/no-coding-required-how-new-designers-are-using-gamemaker-to-create-indie-smash-hits/ (visitado 21-05-2015).

[17] Pradeep. *MonoGame Project No Longer Requires XNA Framework.* URL: http://microsoft-news.com/monogame-project-no-longer-requires-xna-framework/ (visitado 29-05-2015).

[18] Raúl Rosso. *Cuatro entornos de desarrollo para crear videojuegos sin tener que programar.* URL: http://blog.uptodown.com/cuatro-entornos-de-desarrollo-para-crear-videojuegos-sin-saber-programar/ (visitado 10-06-2015).

[19] Mike Seymour. *CryEngine 3 vs Unreal Engine 4.* URL: http://www.fxguide.com/quicktakes/cryengine-3-vs-unreal-engine-4/ (visitado 27-05-2015).

[20] Daniel Sidhion. *Review: Construct 2, a Drag and Drop HTML5 Game Maker.* URL: http://code.tutsplus.com/articles/review-construct-2-a-drag-and-drop-html5-game-maker--active-10825 (visitado 06-09-2015).

[21] Chris Stokel-Walker. *¿Qué es exactamente la teoría de juegos?* URL: http://www.bbc.com/mundo/noticias/2015/02/150220_teoria_de_juegos_que_es_finde_dv (visitado 11-09-2015).

[22] *Torque 3d: Has anyone used it since it went open source, or in the past year? Experiences?* URL: http://www.reddit.com/r/gamedev/comments/1c9gvx/torque_3d_has_anyone_used_it_since_it_went_open/ (visitado 10-06-2015).

[23] Biografías y Vidas. *John Von Neumann.* URL: http://www.biografiasyvidas.com/biografia/n/neumann.htm (visitado 11-09-2015).

[24] Wikipedia. *List of Unreal Engine games.* URL: http://en.wikipedia.org/wiki/List_of_Unreal_Engine_games (visitado 25-05-2015).

[25] la Enciclopedia Libre Wikipedia. *Teoría de Grafos.* URL: https://es.wikipedia.org/wiki/Teor%C3%83%C2%ADa_de_grafos (visitado 11-09-2015).