

Desarrollo de la Inteligencia Artificial para un Videojuego de Estrategia en Tiempo Real



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:

Jorge Quesada Mora

Tutor/es:

Francisco José Gallego Durán

Septiembre 2016



1. JUSTIFICACIÓN Y OBJETIVOS

Todavía puedo recordar cómo veía de pequeño jugar a mi hermano en nuestro viejo PC jugando al Starcraft. Yo solo quería ver, a mí no me interesaba jugar. Con el paso de los años eso cambio. Jugué, y mucho, a muchos juegos distintos, pero ninguno igualó a ese Starcraft de hacia tantos años. La pasión por los juegos de estrategia y por la programación me llevo a plantearme el tema de este trabajo, quería programar algo parecido a lo que yo jugaba, así que decidí embarcarme en esta odisea: El desarrollo de una Inteligencia Artificial para un videojuego de Estrategia a Tiempo Real.

Los objetivos, en un principio, parecían sencillos: crear una inteligencia artificial de un juego de estrategia que jugara contra otra de su misma clase. Que cada partida fuera distinta, que el observador pueda ver las decisiones que toma y porque las toma. A su vez aprender sobre los distintos juegos de estrategia que hay, en que se diferencian unos de otros, y elegir el tipo que más se adaptara a mis necesidades. Pero sobretodo, el principal objetivo era aprender, crear desde cero un producto por mí mismo del que me sienta orgulloso y conseguir resolver los muchos problemas que se me plantearían a lo largo del desarrollo. El más grande de todos, y que para ser sinceros no lo había pensado en un principio, era que, para crear una Inteligencia Artificial de un juego de estrategia, necesitas un juego de estrategia.

1.1.OBJETIVOS

A continuación, se expondrán los distintos objetivos que se definieron para este Trabajo de fin de Grado:

- Repaso de la historia y tipos de videojuegos de estrategia.

- Estudio de los tipos y modelos de Inteligencia Artificial en videojuegos *RTS* existentes.
- Análisis y selección de las mecánicas básicas más interesantes de los distintos videojuegos *RTS* existentes.
- Implementación de un motor sencillo de videojuego *RTS* preparado para experimentar con Inteligencia Artificial.
- Implementación de un sistema de depuración del videojuego en tiempo real con una interfaz de introspección y control del estado interno.
- Implementación y experimentación con distintas técnicas de Inteligencia Artificial seleccionadas.
- Experimentación con las distintas técnicas implementadas y obtención de conclusiones sobre desarrollo y aplicación genérica a distintos videojuegos presentes y futuros.

Dar gracias a mi tutor Fran por haberme ayudado a crear algo con lo que soñaba desde pequeño. Y gracias a Lluís, mi profesor de matemáticas del instituto, por haberme enseñado a amar las ciencias y a convertirme en la persona que soy hoy en día.

A mi familia, por haber aguantado a este cabezota tantos años. A mi hermano, por haber sido el mejor amigo, mentor y ejemplo a seguir que he tenido. A Anna por haberme ayudado en todo momento. A Pablo y Javier, por haberme acompañado en los mejores años de mi vida.

"Dudo que el ordenador llegue algún día a igualar la intuición y capacidad creativa del sobresaliente intelecto humano."

—Isaac Asimov

2. INDICE DE PAGINAS

1.	JUSTIFICACIÓN Y OBJETIVOS.....	1
1.1.	OBJETIVOS	1
2.	INDICE DE PAGINAS	6
3.	INDICE DE FIGURAS	9
4.	INTRODUCCIÓN	11
5.	MARCO TEÓRICO.....	12
5.1.	Los juegos de Estrategia.....	12
5.1.1.	Turn Base Strategy (<i>TBS</i>).....	13
5.1.2.	Turn Base Tactics (<i>TBT</i>)	14
5.1.3.	Real Time Strategy (<i>RTS</i>).....	15
5.1.4.	Real Time Tactics (<i>RTT</i>)	16
5.1.5.	MMORTS	16
5.1.6.	MOBA	16
5.2.	Inteligencia Artificial	17
5.2.1.	IA en distintos tipos de juegos	17
5.2.2.	IA en juegos de Estrategia.....	18
5.3.	Motores de Videojuegos	19
5.3.1.	Unreal Engine	19
5.3.2.	Unity	20
5.4.	Una partida en un RTS	21
6.	METODOLOGÍA.....	24
7.	PLANIFICACIÓN.....	26

8. TRABAJO REALIZADO	27
8.1. ¿Por qué RTS?	27
8.2. ¿Por qué Unity?	27
8.3. El juego	27
8.4. Distribución de las clases	29
8.5. Unidades.....	30
8.6. Edificios.....	32
8.7. Funcionalidades	34
8.7.1. Información de las entidades.....	34
8.7.2. Tareas	34
8.7.3. Lista de tareas.....	35
8.7.4. Mapa de valores de construcción	36
8.7.5. Algoritmo de mapeado	38
8.7.6. Distribución de gastos.....	41
8.7.7. Perfiles.....	42
8.7.8. Escuadras.....	44
8.7.9. Parser	46
8.8. Inteligencia Artificial	46
8.8.1. ¿Qué construir?	46
8.8.2. ¿Dónde construir?	49
8.8.3. ¿Qué unidades hacer?	52
8.8.4. ¿Cuántas unidades hacer?	53

8.8.5.	¿Cuándo hacer las unidades?.....	53
8.8.6.	¿Cómo explorar?.....	54
8.8.7.	¿Dónde atacar?.....	55
8.8.8.	¿Cuándo atacar?	56
8.8.9.	¿Con qué atacar?.....	56
8.9.	Sistema de depuración visual.....	57
8.9.1.	Gizmos.....	58
8.9.2.	Niebla de guerra (FOW)	59
8.9.3.	Interfaz	60
9.	CONCLUSIONES	66
10.	GLOSARIO	67
10.1.	Definiciones	67
10.2.	Abreviaturas	70
11.	BIBLIOGRAFÍA Y REFERENCIAS.....	71
12.	ANEXOS.....	Error! Bookmark not defined.

3. INDICE DE FIGURAS

Ilustración 1: Command and Conquer, 1995	11
Ilustración 2: Civilization V	13
Ilustración 3: XCOM Enemy Unknown.....	14
Ilustración 4: Starcraft 1	15
Ilustración 5: Dota 2	17
Ilustración 6: Interfaz de Unity	20
Ilustración 7: La unidad azul sí que ve a la unidad roja ya que se encuentra en una altura superior, en cambio la roja no sabe dónde se encuentra la azul.....	22
Ilustración 8: Unidades del equipo rojo.....	31
Ilustración 9: Edificios de la IA roja.....	32
Ilustración 10: Ejemplo de tarea mostrada en tiempo de ejecución.....	35
Ilustración 11: Lista de tareas que se muestra en la interfaz del juego.....	36
Ilustración 12: Captura del mapa de valores de construcción, donde se aprecian los distintos tipos de terrenos.....	37
Ilustración 13: Mapa de valores de construcción donde se observan los distintos colores que representan distintos valores.	40
Ilustración 14: Ejemplo de perfil, archivo Aggressive.dataUser	43
Ilustración 15: Escuadra poniendose en posición	44
Ilustración 16: Escuadra ordenada con todo tipo de unidades.	45
Ilustración 17: Supply	47
Ilustración 18: Barracs.....	47
Ilustración 19: Factory.....	47
Ilustración 20: Turret.....	48
Ilustración 21: Academy	48
Ilustración 22: Main Building.....	48

Ilustración 23: Edificios colocados mediante el algoritmo de construcción.	50
Ilustración 24: El jugador Rojo construye sus torretas en los dos lados del acantilado porque tiene dos rampas. En cambio, el Azul solo en uno solo, porque en el otro lado no hay rampa que defender.....	51
Ilustración 25: Explorador de la IA azul obteniendo datos de la IA roja. ..	54
Ilustración 26: Batalla entre las dos facciones, la roja cuenta con la ventaja defensiva.....	56
Ilustración 27: IA roja retirandose a base con las pocas fuerzas restantes tras un ataque fallido.	57
Ilustración 28: Builder trazando el camino del Main Building al mineral. .	58
Ilustración 29: Gizmos del radio de visión (azul) y del radio de ataque (rojo). Encima de las unidades se pueden apreciar las barras de vida.....	58
Ilustración 30: Zona del mapa donde se aprecian los tres estados de la niebla de guerra.....	59
Ilustración 31: Interfaz – cuadros superiores.....	61
Ilustración 32: Interfaz - consola de mensajes.....	61
Ilustración 33: Interfaz - zona del minimapa.	62
Ilustración 34: Interfaz - cuadro de informacion, pestaña "Info Selection".	63
Ilustración 35: Interfaz - cuadro de informacion, pestaña "Tasks".....	64
Ilustración 36: Interfaz - cuadro de informacion, pestaña "Buildings".....	64
Ilustración 37: Interfaz - cuadro de informacion, pestaña "Units".....	65
Ilustración 38: Interfaz - cuadro de informacion, pestaña "Enemy Info" . .	65

4. INTRODUCCIÓN

Hace ya más de 30 años surgió uno de los primeros videojuegos que daría pie a uno de los más famosos y exitosos géneros de los videojuegos. Estamos hablando del juego **Ancient Art of War** (Evryware, 1984), que fue la punta de lanza de lo que hoy se conoce como Real-Time Strategy¹ (*RTS*²) o Juegos de estrategia a tiempo real. Género que ha visto nacer a obras de arte como la saga **Command and Conquer** (Westwood Studios), la saga **Starcraft** (Blizzard), la saga **Warcraft** (Blizzard) o la saga **Age of Empires** (Ensemble Studios), entre muchos otros. Desde ese año, hemos podido observar como la industria del videojuego ha avanzado a pasos agigantados, pero la esencia de estos juegos sigue siendo la misma, gestiona mejor tus recursos y toma mejores decisiones para imponerte a tu oponente.



Ilustración 1: Command and Conquer, 1995

¹ Se utilizarán gran cantidad de abreviaturas en este documento. La primera vez que cada abreviatura sea utilizada se pondrá junto a la palabra al completo. Además, final del trabajo en el apartado de glosario se puede buscar el significado de cada abreviatura para más comodidad.

² Los elementos que se encuentren en cursiva serán elementos disponibles en el glosario al final del documento para poder leer su definición.

Por otro lado, tenemos la Inteligencia Artificial (*IA*), que ha formado parte de los videojuegos desde hace décadas. Un oponente virtual al que batir, que te suponga un reto pero que a su vez no frustre al jugador hasta el punto de dejar de jugar. Esa ha sido siempre la preocupación de los programadores, encontrar ese punto perfecto, donde los dos platos de la balanza estén en equilibrio. El principal problema de la mayoría de las *IA*, hoy en día, es que no basan sus decisiones en los datos obtenidos por ellas mismas, sino que los basan en los datos del jugador, datos a los que tiene acceso todo el tiempo. Es decir, las *IA* tienen como objetivo ser buenos oponentes, no ser oponentes realistas.

Y también en equilibrio entre dos platos de la balanza se encuentra este Trabajo de fin de Grado (*TFG*). Tenemos, por una parte, una Inteligencia Artificial que se cimienta sobre un juego de estrategia, y por otra, la imperante necesidad de mostrar correctamente los datos sobre los que se basan las decisiones de esta *IA* ya que nosotros seremos meros observadores.

5. MARCO TEÓRICO

5.1. Los juegos de Estrategia

El género de los juegos de estrategia es tan extenso que tenemos que subdividirlo en una gran cantidad de subgéneros para poder abarcar la gran diversidad de títulos que lo engloban. Todos ellos tienen algo en común, la esencia de los juegos de estrategia: hay que gestionar correctamente los recursos a los que uno tiene acceso y planear correctamente las jugadas para poder sobreponernos al oponente.

5.1.1. Turn Base Strategy (TBS)

Los TBS o los juegos de estrategia por turnos, como su nombre indica, son los más parecidos a sus primos lejanos, los juegos de estrategia de mesa. En ellos, cada jugador tendrá un turno para realizar sus movimientos. En el cual, el o los oponentes no podrán realizar ninguna otra acción. De esta forma se les da a los jugadores mucho más tiempo para pensar sus jugadas permitiéndoles pensar más allá del turno que les ocupa. Es, al ser basado en turnos, mucho más lento a la hora de jugarlo que otros tipos de juegos de estrategia a tiempo real.



Ilustración 2: Civilization V

A su vez se subdivide en varios tipos como: Turn Base Tactics (TBT), construcción de imperios o Semi-TBS. Todos tienen distinciones:

- TBT: que no tengas que gestionar tus recursos sino únicamente tus unidades. Con juegos tan famosos como los **XCOM** (Firaxis Games, 2013).

- Construcción de Imperios: que tengas que manejar un imperio de forma militar, política, económica e incluso religiosa, como la saga **Civilization** (MicroProse)
- Semi-TBS: juegos que combinan los *RTS* y los *TBS* como la saga **Total War** (The Creative Assembly).

5.1.2. Turn Base Tactics (TBT)

Considerándose, a su vez, un subgénero dentro de los *TBS* se diferencia lo suficiente como para poder separarlo de este. En este tipo de juegos no tendremos que gestionar recursos económicos como tal en el campo de batalla. Centrándonos así, en las decisiones tácticas del campo de batalla. Normalmente la cantidad de unidades que utilizaremos será relativamente pequeña comparada con otros juegos del género. En este subgénero englobamos juegos como los **XCOM**, los **Fire Emblem** (Intelligent Systems) o los **Final Fantasy Tactics** (Squaresoft, 1997).



Ilustración 3: XCOM Enemy Unknown

5.1.3. Real Time Strategy (RTS)

El subgénero de los juegos de estrategia por excelencia. En él nos enfrentaremos a nuestro oponente a tiempo real. Nuestras decisiones se verán reflejadas automáticamente en la partida. Tiene un ritmo mucho más frenético que el de los *TBS*. Tiene como objetivo gestionar los recursos obtenidos, unidades y edificios para superar al oponente. Suele haber varios tipos de recursos a lo largo del mapa, los cuales tendremos que recolectar para poder crear tanto unidades como vehículos o mejoras.

Según la distribución de los recursos podríamos encontrar dos principales tipos de juego. En primer lugar, los que tienen los recursos en el punto de inicio de la partida, con determinados nodos de recursos donde construiremos expansiones. Frente a este tenemos el caso contrario, donde todo el mapa está lleno de todo tipo de recursos y podremos crear nuestras expansiones donde queramos. Ejemplos de estos tipos de juegos son el **Starcraft** y el **Age of Empires** respectivamente.



Ilustración 4: Starcraft 1

Este tipo de juegos de estrategia junto con los Multiplayer Online Battle Arena (*MOBA*) son los más propensos a tener una rama competitiva, ya que, al ser a tiempo real son mucho más dinámicos que otros tipos de juegos.

5.1.4. Real Time Tactics (*RTT*)

Al igual que los *TBT* de los *TBS*, los Real Time Tactics se pueden considerar un subgénero de los *RTS*, donde el jugador deberá gestionar sus unidades como principal recurso, ya que, en este tipo de juegos no se suelen poder crear ni más unidades ni edificios.

5.1.5. MMORTS

Masive Multiplayer Online Real-Time Strategy (*MMORTS*) se refieren a los juegos de estrategia a tiempo real donde cientos de jugadores, sino miles, son capaces de jugar al mismo momento. Debido a la naturaleza de estos juegos suelen estar más orientados a juegos de navegador de gestión (*Travian*, *O-Game*, etc.). O juegos de plataformas móviles (*Clash of Clans*, *Star Wars: Commander*, etc.). Son juegos orientados a partidas persistentes, de meses o años de duración, al contrario de los *RTS* clásicos.

5.1.6. MOBA

Multiplayer Online Battle Arena o *MOBA* como se conoce más comúnmente. En este tipo de juego se enfrentarán dos equipos formados por un reducido número de jugadores que normalmente manejarán no más de una sola unidad o héroe (dada la importancia de este). Son juegos orientados, casi en su totalidad, al juego online con un importantísimo peso en su rama competitiva.

Algunos de los juegos competitivos, o *e-sports*, más famosos pertenecen a este género, como son el **League of Legends (LoL)** o el **Defense of the Ancient (DotA)**. Este género de juego surgió a partir de mods creados para otro tipo de juegos. Este es el caso de uno de los primeros y más famosos, el *DotA*, que era un mapa personalizado del **Warcraft 3**.



Ilustración 5: Dota 2

5.2. Inteligencia Artificial

5.2.1. IA en distintos tipos de juegos

La Inteligencia Artificial es el oponente virtual contra el que nos enfrentaremos a la hora de jugar. Como jugadores, queremos que sea un oponente justo, a nuestro nivel. Un oponente que nos suponga un reto, pero que no nos llegue a frustrar (a menos que ese sea el objetivo del juego). Como programadores tenemos que conseguir ese objetivo. Por lo tanto, en muchísimos juegos, la Inteligencia Artificial no tiene como objetivo ser perfecta, sino lo anteriormente mencionado, que esa máquina

sea un buen oponente, parecerse lo máximo posible a una persona, y una persona no va a ser perfecta jugando

A la hora de desarrollar una IA tenemos que saber qué cosas tiene que saber hacer y qué cosas no tiene que saber hacer. Por ejemplo, en un videojuego de fútbol, sería muy fácil hacer que una IA metiera gol siempre, tiene todos los datos a su alcance, es un simple cálculo. Pero esto no proporcionaría una buena experiencia. O en un juego de estrategia, la IA podría hacer la gestión perfecta de los recursos, podría *micrear* las unidades de forma perfecta, pero eso no es realista. Así que, en muchos juegos la IA en vez de pensar por sí misma, basa sus decisiones en las decisiones del jugador para igualar la situación.

5.2.2. IA en juegos de Estrategia

En los juegos de estrategia se tienen que tomar gran cantidad de decisiones y se realizan gran cantidad de acciones de diferente tipo. Para programar todo lo anterior de forma correcta, separaremos el código general en partes más pequeñas, partes de las cuales los *managers* se encargarán de ejecutar.

La construcción de los edificios, la creación de unidades, el ataque al oponente, la exploración del mapa, etc. todas ellas son distintas tareas que una IA tiene que ser capaz de llevar a cabo. Pero a su vez, todas ellas serán realizadas por diferentes *managers*. Esto es lo que hace la IA tan potente, el nivel de abstracción tiene que ser muy alto para poder realizar esta gran cantidad de tareas de forma correcta. Pese a todo, estos *managers* estarán interconectados unos con otros para poder comunicarse en caso de ser necesario.

5.3. Motores de Videojuegos

Los motores de videojuegos permiten hoy en día el desarrollo de videojuegos sin la necesidad de que cada empresa tenga que crear su propio motor. La nueva tendencia de proporcionar precios asequibles a los desarrolladores independientes y a las pequeñas empresas ha permitido un “bum” en el mercado de los videojuegos. De esta forma, ha hecho posible el acceso para estos desarrolladores a herramientas anteriormente imposibles de conseguir para ellos. Los motores más famosos tienen diferentes tarifas para todo tipo de perfiles en el sector. Desde gratuitos, para desarrolladores independientes o equipos pequeños (donde la empresa del motor se lleva un porcentaje de las ganancias del juego) hasta versiones pro con mejores características, pero no al alcance de todos los bolsillos. A continuación, se describirán un par de los motores más famosos del sector, y más adelante se argumentará porque se ha elegido uno u otro para la realización de este *TFG*.

5.3.1. Unreal Engine

Motor creado por Epic Games, fue creado inicialmente en el año 1998 para el juego Unreal, y utilizado en varios de sus posteriores proyectos. Pero este motor no ha sido utilizado solo por su propia empresa, sino que muchos de los más conocidos videojuegos han sido desarrollados en este motor, desde triple A hasta *indies*. Este motor no solo permite el desarrollo de juegos para Pc, sino también juegos de consolas. La política actual del motor te permite desarrollar con él de forma gratuita, y solo en caso de comercializar tu producto y ganar más de 3000\$ con el deberías pagar a Epic el 5% de las ganancias.

5.3.2. Unity

Motor desarrollado por la empresa Unity Technologies, al igual que Unreal es un motor multiplataforma permitiendo el desarrollo para cualquier plataforma, desde *PC* hasta móviles y consolas. Fue uno de los primeros motores en empezar la política de dejar a los desarrolladores el acceso de forma gratuita. Cuenta con varias versiones desde la Personal, totalmente gratuita, a la Pro, de pago. Respecto a la política de pagos, Unity difiere un poco de Unreal. Los desarrolladores que tengan más de 100 mil \$ de ingresos anuales no podrán usar la versión Personal y conforme más ingresos tenga el desarrollador estará obligado a usar una versión más completa del motor.

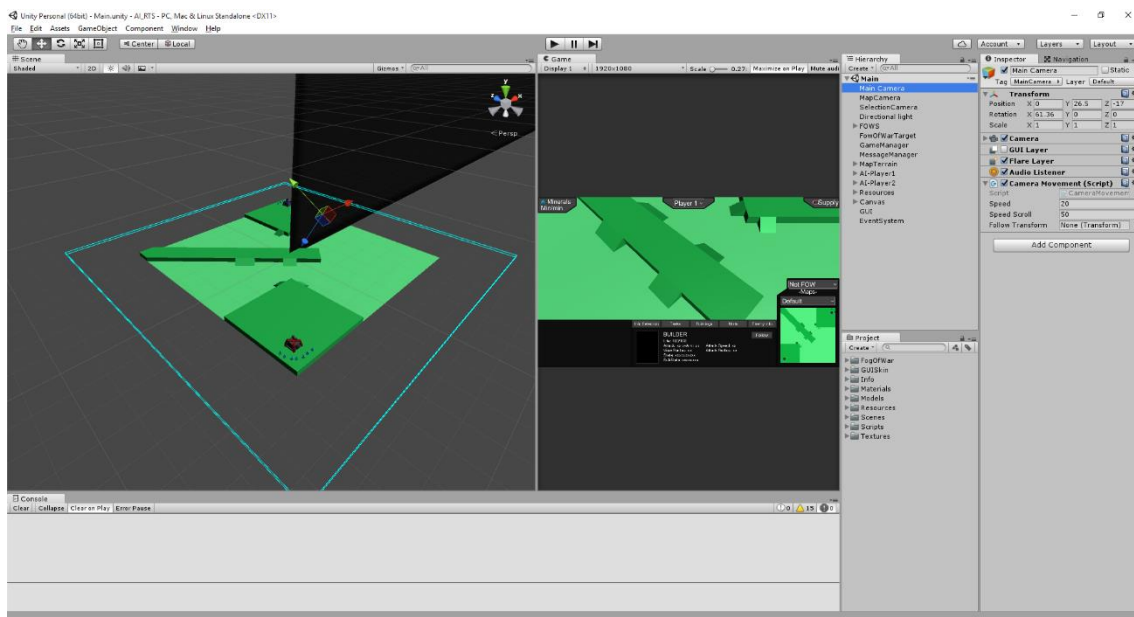


Ilustración 6: Interfaz de Unity

5.4. Una partida en un RTS

Dado que se van a nombrar muchos conceptos y situaciones que para alguien que no esté muy versado en los *RTS* serán difíciles de comprender, a continuación, se va a explicar cómo será una partida en un *RTS* y en concreto en este.

Para empezar, explicar un poco el objetivo de la partida. Una partida, constará de dos jugadores. Estos jugadores empezarán la partida en posiciones simétricas en un mapa también simétrico. Con esto se consigue que ninguno de los dos tenga ventaja de terreno. El objetivo de cada jugador es acabar con el oponente, y en este juego en concreto, con su edificio principal.

Dado que los dos jugadores tienen las mismas posibilidades en el inicio de la partida (posiciones simétricas), lo que hará que gane uno u otro serán las decisiones que tome a lo largo de esta. Podríamos comparar esto con una partida de ajedrez, al inicio los dos jugadores tienen las mismas posibilidades, pero son los movimientos de las piezas lo que inclina la balanza.

Cada jugador dispone al inicio de partida de un *Main Building* y cinco *Builders*, además de siete menas de mineral cercanas al edificio central de las que se obtienen los recursos. Desde este momento, el jugador ha de saber si va a tomar un enfoque más defensivo o agresivo, ya que de esto dependerán las decisiones que tome.

Por otro lado, tenemos el terreno. Como se ha dicho, el terreno es simétrico por el centro del mapa, para que el terreno no sea influyente desde el inicio de la partida. Pero un buen uso del mapa puede decantar la balanza a tu favor. Una de las mecánicas básicas en muchos *RTS* éste incluido, son las diferentes alturas del terreno. Este se divide en varios niveles de altura, conectados mediante rampas. Estas rampas son los únicos puentes de conexión entre los niveles, lo que los convierte en un cuello de botella, puntos estratégicos en el mapa, difíciles de atacar y fáciles de defender. La altura también influye en la visión de las unidades y edificios. Esta visión abarca su altura y todas las inferiores, pero no podemos ser capaces de ver por encima de los saltos de altura. Esta mecánica da ventaja a las unidades que estén a más altura.

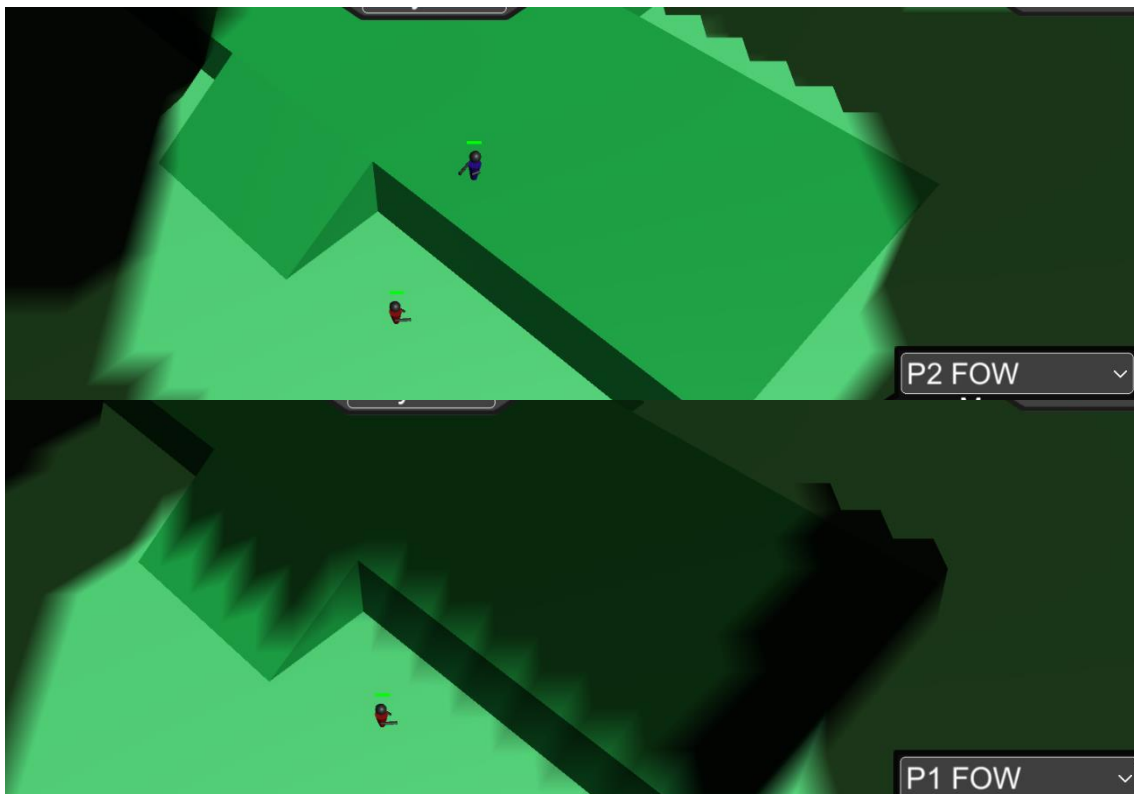


Ilustración 7: La unidad azul sí que ve a la unidad roja ya que se encuentra en una altura superior, en cambio la roja no sabe dónde se encuentra la azul.

Los suministros representan la cantidad de unidades que un jugador puede crear. Es un recurso más que hay que tener en cuenta en una partida, si nos sobran muchos suministros hemos desperdiciado minerales en crearlos cuando podíamos haberlos usado en otras tareas. En cambio, si tenemos muy pocos, pueden bloquear la producción de unidades, ya que no podemos tener más unidades que la cantidad que nos permiten los suministros. Son por tanto un edificio clave, atacarlo y destruirlo nos permitirá que el enemigo se quede bloqueado sin poder hacer unidades hasta que los vuelva a construir, este término, se conoce como *Supply Block*.

En cuanto a las unidades, podríamos definir un combate entre unidades como una partida al “piedra, papel, tijera”. Una unidad es a su vez, muy efectiva contra un tipo y muy poco efectiva contra otr. Así que el posicionamiento y la cantidad de unidades de cada tipo que tienes es algo importante, te permite tener una ventaja táctica sobre tu oponente, siendo capaz de vencerle pese a tener menos unidades. Por ejemplo, si yo tengo cinco “piedras” pero el oponente tiene diez “tijeras” lo más probable es que gane pese a estar en desventaja numérica.

6. METODOLOGÍA

La primera fase del *TFG* ha consistido en la búsqueda de información en los libros sobre Inteligencia Artificial que he podido encontrar. Algunos de estos libros son el ***Game Programming Wisdom***³, ***Programming Game AI by Example*** o el ***Game AI PRO***. En ellos he podido aprender gran cantidad de técnicas bastante útiles para el desarrollo de una buena *IA*. Lo malo es que ejemplos prácticos de la *IA* de un *RTS*, que es el tipo de juego que a mí me interesaba, no he encontrado ninguno. Así que, sabiendo los conceptos teóricos que he aprendido de los libros, me planteé una serie de elementos que el producto final debería de tener:

- La capacidad de construir edificios, sabiendo cuándo y dónde colocarlos. Para un ser humano es fácil tomar este tipo de decisiones prácticamente de forma intuitiva, pero una máquina necesita datos que comparar para tomar las decisiones.
- Crear unidades sabiendo que cantidad y cuando crearlas. Es decir, que unidad es mejor en cada momento y cuantas se han de crear con los recursos de los que se disponen.
- Poder atacar y defenderse, explorar el mapa o decidir el momento de ataque y el lugar. Los ataques serán más o menos efectivos dependiendo de la información que tengamos del oponente y del momento del ataque.

Para conseguir dichos objetivos, todo el desarrollo de la *IA* se ha basado en responder a una serie de preguntas:

- ¿Qué construir?

³ En el apartado Bibliografía hay referencias completas a dichos libros.

- ¿Dónde construir?
- ¿Cuándo construir?
- ¿Qué unidades hace?
- ¿Cuántas unidades hacer?
- ¿Cuándo hacer las unidades?
- ¿Cómo explorar?
- ¿Dónde atacar?
- ¿Cuándo atacar?
- ¿Con que atacar?

Una vez respondidas estas preguntas ya se podría desarrollar la IA. Antes de eso, hay que programar la base del juego de estrategia, porque sin juego, no hay IA. Y, por último, se implementaría la interfaz que mostrará a los observadores todos los datos.

Respecto a la hora de distribuir el trabajo, se iba complementando tanto el responder a estas preguntas como la implementación de forma que el trabajo no se volviera monótono.

7. PLANIFICACIÓN

El proyecto se planificó para realizarlo en dos meses de trabajo. Teniendo tanto julio como agosto para realizar el proyecto. Además de la semana inicial de septiembre para pulir y finalizar los últimos detalles.

Sin fines de semana son 40 días laborables, siendo un proyecto de 300 horas daba 7.5 horas diarias. Preferí trabajar fines de semana antes de tener que realizar tantas horas diarias. Así que al final, se repartieron 5.5 horas diarias entre semana, más 10 horas en el fin de semana. Eso hace un total de 37.5 horas semanales, multiplicado por las 8 semanas de julio y agosto dan las 300 horas. En caso de necesitar más horas, el horario de agosto sería más ajustado.

Han surgido algunos problemas a lo largo del desarrollo, desde más carga de trabajo de la esperada, hasta momentos en los que no sabía cómo realizar diferentes tareas. El problema principal, anteriormente mencionado, ha sido tener que realizar la base del *RTS*, donde se han perdido unas preciadas horas de trabajo que no se han utilizado para la programación de la *IA* o de la interfaz.

La solución de estos problemas no ha ido más allá de más horas de trabajo para realizar alguna tarea. También tuve que leer varios libros de *IA* para saber cómo hacer bien la distribución de los *managers*. Y para los momentos de bloqueo, lo que más me ha servido era libreta y boli e intentar apuntar las posibles soluciones al problema y a partir de ahí confeccionar una solución final. Lo que sí se puede decir es que no ha habido grandes problemas en el desarrollo que lo bloqueen mucho tiempo, pero si varios problemas menores.

8. TRABAJO REALIZADO

8.1. ¿Por qué RTS?

El haber elegido el género de los *RTS* como base para crear la inteligencia artificial tiene varias razones. La primera de ellas es mi claro favoritismo a este tipo de juegos, ya que son los que más he jugado y disfrutado desde que tengo uso de razón. La segunda de ellas es que nunca he encontrado una *IA* en este tipo de juegos que se sienta como un jugador “real”. Las partidas con este tipo de *IA* se hacían bastante predecibles, así que mi objetivo era evitar eso, que la *IA* se adaptase a lo que iba ocurriendo.

8.2. ¿Por qué Unity?

La elección del motor a la hora de crear un videojuego es una importante elección. Pese a que Unreal es mucho más potente que Unity en algunos aspectos, al final me decante por Unity por las siguientes razones. El tiempo de aprendizaje que llevaría aprender Unreal a mi nivel actual de Unity sería demasiado alto. El sistema de scripts de Unity era perfecto para mis propósitos, y esto, sumado a la experiencia anterior que tenía con este motor hizo que no fuera una decisión difícil de tomar.

8.3. El juego

Antes de realizar la *IA* se diseñó y preparó el juego sobre el que se implantaría. Aun así, el juego ha sufrido bastantes cambios a lo largo del desarrollo para adaptarse a las necesidades que iban surgiendo. Dado que lo importante del proyecto es la Inteligencia Artificial y la forma de mostrar los datos al observador, se decidió crear un juego bastante sencillo.

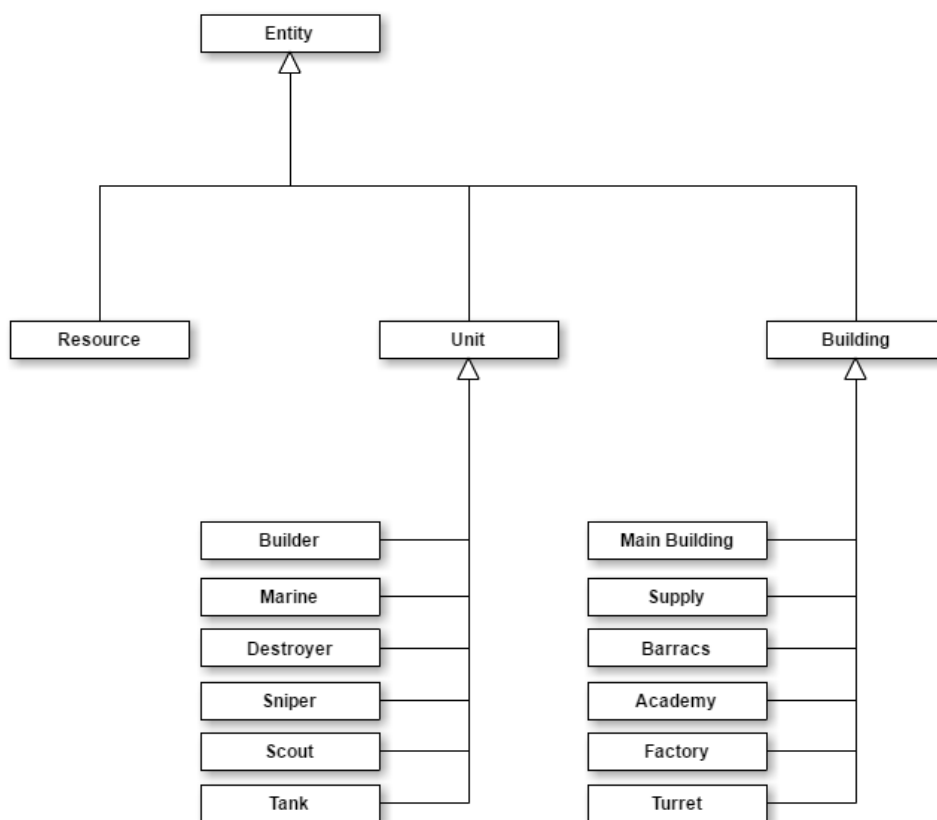
El juego consta de un solo recurso, los minerales. Esto es fácilmente ampliable a otros tipos de recursos, pero así los esfuerzos se podían centrar en las cosas importantes.

Por otro lado, tenemos los edificios y unidades. Hay un total de 6 edificios: el *Main Building*, *Barracs*, *Factory*, *Supply*, *Academy* y *Turret*. También tenemos 6 tipos de unidades distintas: *Builder*, *Marine*, *Destroyer*, *Sniper*, *Scout* y *Tank*.

El mapa es sencillo, con dos diferentes alturas (ampliable a las que sean necesarias). La línea de visión no puede saltar estas alturas, pero si estás arriba sí que ves las unidades que hay abajo, dando así ventaja a las posiciones superiores.

La situación de las bases al inicio de la partida es simétrica respecto al centro del mapa, tenemos cada base de cada jugador en las esquinas contrarias, con 7 menas de mineral en cada una de las bases. Las bases se encuentran en elevaciones para que sean más fácilmente defendibles.

8.4. Distribución de las clases



8.5. Unidades

Cada unidad tiene el mismo tipo de variables. Lo que hace única a cada una de ellas son los valores de dichas variables. El ataque, velocidad, radio de visión, etc. Son algunas de las muchas variables que editándolas creamos unidades totalmente nuevas. Para hacer este proceso más sencillo se han abstraído todos los datos del código a archivos externos. La obtención de dichos datos se hace mediante el *parseador*⁴.

Los datos sobre las unidades son cargados desde ficheros fuera de Unity, ficheros creados únicamente con este propósito, con la extensión “.dataUnit” en ellos tendremos los siguientes datos:

	Builder	Marine	Destroyer	Sniper	Scout	Tank
Cost	50	50	125	150	100	225
Priority	2	2	2	2	2	2
Created In ⁵	MB	B	B	B	F	F
Creation Time	10	15	20	25	10	30
Buildings Needed	MB	MB, B	MB, B	MB, B, A	MB, F	MF, F
Attack Speed	1	1.5	1.5	1.8	1	2
Atk. Dmg. Light	0	10	10	45	10	35
Atk. Dmg. Armored	0	10	40	10	10	35
Spend Type ⁶	Eco	Com	Com	Com	Com	Com

⁴ Ver apartado Parser dentro de Funcionalidades

⁵ Las abreviaturas hacen referencia a los tipos de edificio: Main Building (MB), Supply (S), Barracs (B), Academy (A), Factory (F) y Turret (T).

⁶ Las abreviaturas hacen referencia al tipo de gasto: Economico (Eco) o de Combate (Com).

A estos datos sería muy fácil añadir cualquier otro que no esté en estas listas, por ejemplo, la vida de las unidades, el radio de visión o el de ataque. Por tanto, cada unidad va definida por las siguientes características: coste en minerales, prioridad a la hora de crearse, el edificio donde se crea, la duración para crearla, los edificios necesarios, la velocidad de ataque, el ataque tanto para unidades de a pie como vehículos y edificios y por último el tipo de gasto que es (si económico o de combate).

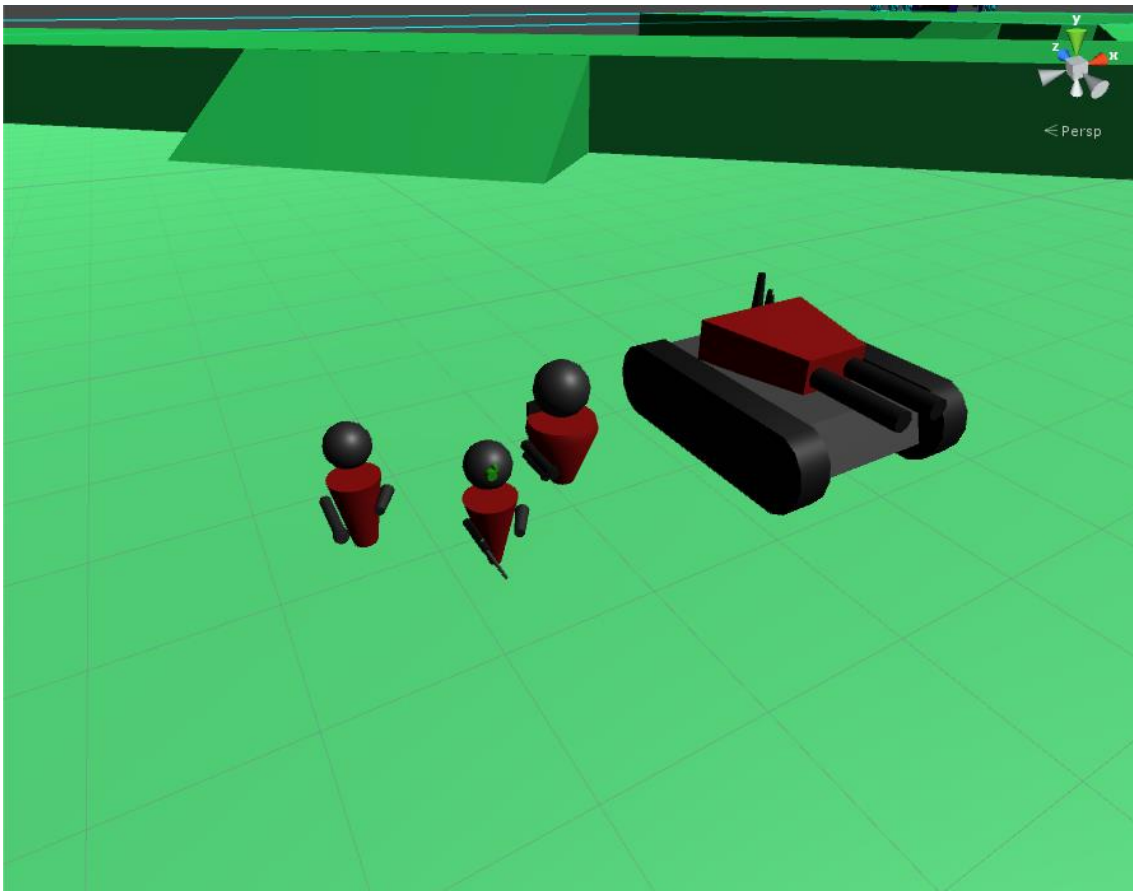


Ilustración 8: Unidades del equipo rojo

Como se explicará más al detalle en los posteriores apartados, las unidades están supeditadas tanto a las órdenes de las escuadras como de la IA. Pero aun así pueden tomar decisiones por sí mismas, siempre que no interfieran en las ordenes que le han mandado. Cada unidad contiene una máquina de estado la cual rige sus acciones. Las unidades de combate y los

constructores tendrán estado comunes, pero también diferentes. Por defecto, todas las unidades comienzan en estado “reposo”. La IA o la escuadra es la que le cambiarán dicho estado, pero en determinadas situaciones reaccionan por sí mismas, como por ejemplo, si una unidad enemiga entra en su campo de visión, le atacan.

Los estados pueden cambiar por sí solos, por ejemplo, si un trabajador termina de construir un edificio pasa de estado “construyendo” a “reposo”, o si está atacando a un enemigo y este muere, pasa de “atacando” a “reposo”.

8.6. Edificios

Al igual que para las unidades, hay 6 tipos de edificios (definidos en el apartado de definiciones). El objetivo, ya que el juego pretendía ser sencillo, es que hubiera edificios de todos los tipos. Hay tanto edificios de producción de unidades (*Main Building*, *Barracs* o *Factory*), infraestructura (*Academy* o *Supply*) y de defensa (*Turrets*). Los datos de estos edificios son sacados de archivos externos para poder cambiarlos en cualquier momento.

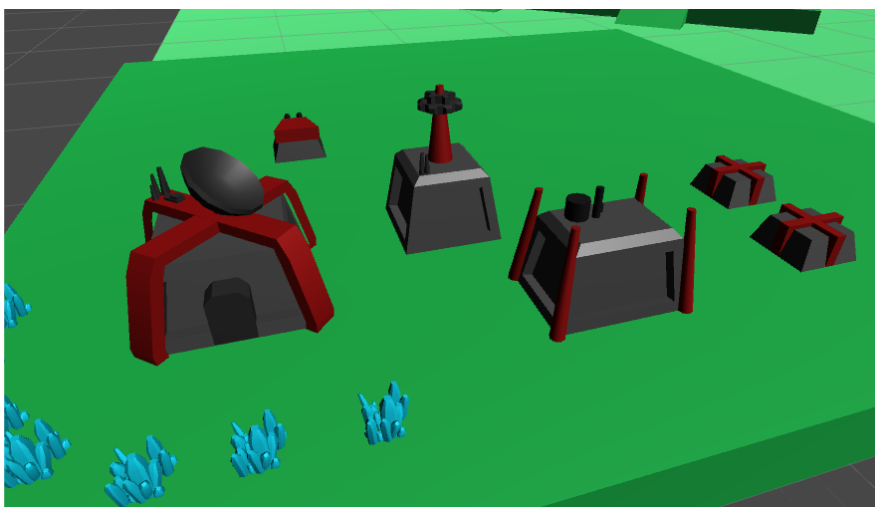


Ilustración 9: Edificios de la IA roja

IA EN UN VIDEOJUEGO RTS

	Main Building	Supply	Barracs	Academy	Factory	Turret
Cost	400	50	125	150	100	225
Size	4x4	2x2	4x3	3x3	4x3	2x2
Priority	5	1	2	2	2	2
Creation Time	60	20	30	45	40	20
Buildings Needed	-	-	S	B	B, A	B
Spend Type	Eco	Eco	Eco	Eco	Eco	Com

Al igual que con las unidades, cualquier variable de los edificios puede añadirse en estos archivos. Estas variables son iguales para cualquiera de las IA. Por otro lado, los edificios al contrario que las unidades, no toman decisiones por si mismos (a excepción de la torreta cuando ataca a enemigos). Ningún edificio creará unidades a menos que le sea mandado desde alguno de los managers.

8.7. Funcionalidades

8.7.1. Información de las entidades

Para cada entidad del juego, como se ha mencionado anteriormente, se lee un archivo externo que contiene datos sobre la misma. Estos datos son recogidos en clases designadas para este propósito. Las “infoClass” contienen los datos más importantes, a los que luego otras partes del código tendrán acceso, por ejemplo, el coste de una unidad, el tamaño de un edificio, que edificios son necesarios tener antes de crear una unidad, etc...


El objetivo de estas clases es que desde todos los puntos del código los datos sobre las entidades del juego sean las mismas. Además, cumple un segundo propósito, la abstracción de los datos del código de forma que sea posible editarlos desde un sencillo archivo de texto.

8.7.2. Tareas

Las tareas se utilizan para guardar la información de las cosas pendientes que tiene que hacer la IA. En ella se guardan varios tipos de datos, primero y más importante, tiene referencia a una “infoClass” que tiene la información sobre la unidad o edificio que se ha de construir. También guarda la prioridad de dicha tarea, cuando más prioritaria sea antes se destinarán los recursos necesarios a crearla. Tiene también guardado el estado actual de la tarea, si está en progreso, en espera o acabada. Por último, tiene una lista de tareas que son necesarias antes de que se pueda llevar a cabo la tarea.

Este último punto es el más importante del sistema de tareas, cuando nosotros añadimos una tarea, si ese edificio o unidad tiene necesidades que

no se han cumplido, se crean nuevas tareas de forma automática para que se cubran estas necesidades. Por ejemplo, se crea la tarea “Crear Marine”, pero para crear un *Marine* necesitamos las *Barracs* (edificio donde se construye), así que, como no tenemos ningunas *Barracs* creamos una tarea para ello. De esta forma, hasta que la tarea “Construir *Barracs*” no se ha completado la de “Crear *Marine*” no se puede completar.



Task 1: Create Builder	Current Status: Waiting
Priority: 16 Tasks Needed: 0	Minerals Needed: 50

Ilustración 10: Ejemplo de tarea mostrada en tiempo de ejecución

8.7.3. Lista de tareas

El *resource manager* tiene una lista de tareas pendientes. En esta lista guardamos tanto las tareas en espera, como las que están en proceso como acabadas. Cada determinado tiempo se hará una limpieza de las tareas acabadas. Si alguna tarea dependía de esta será notificada y la borrará de su lista de tareas necesarias.

Los recursos que se tienen disponibles se destinarán a la tarea que este al inicio de la lista. La prioridad que se le da a las tareas es la siguiente:

- Las tareas en proceso o acabadas van al final de la lista, ya que solo están esperando a ser borradas.
- Las tareas que no dependan de otras van al principio.
- Las tareas que tengan más prioridad van al principio.

Con este sistema, la primera tarea será, una que no dependa de ninguna otra, que no se haya llevado a cabo ya, y que tenga una alta prioridad.

Info Selection	Tasks	Buildings	Units	Enemy Info
Task 1: Create Builder		Current Status: Waiting		
Priority: 17	Tasks Needed: 0	Minerals Needed: 50		
Task 2: Create Destroyer		Current Status: Waiting		
Priority: 8	Tasks Needed: 0	Minerals Needed: 125		
Task 3: Create Scout		Current Status: Waiting		
Priority: 3	Tasks Needed: 0	Minerals Needed: 100		

Ilustración 11: Lista de tareas que se muestra en la interfaz del juego

Pero el sistema de prioridades tenía un fallo. Si se añadía una tarea con poca prioridad y luego se iban añadiendo tareas con más prioridad, la que tenía baja prioridad nunca se le destinarían recursos. Para solucionar el problema, se hizo que cada vez que una tarea se lleva a cabo, el resto de tareas de la lista aumentan en 1 su prioridad. De esta forma, cuando más tiempo estas en la lista, más probable es que seas el primero. Otra de las excepciones que se hizo es con los suministros. Se podía dar el caso que la IA se quede sin suministros pero que haya tareas con más prioridad, haciendo que se bloqueara hasta que se vacíe la lista de tareas. Para ello si la IA se queda sin suministros (la cantidad de unidades que tiene es igual al máximo de suministros) se le dará prioridad inmediata a la tarea de crear suministros, poniéndola la primera.

8.7.4. Mapa de valores de construcción

A la hora de construir un edificio, se necesita saber dónde se va a construir. Dado que no se va a elegir una posición al azar para colocarlo, la IA tiene que tener algún tipo de datos para decidir donde es la mejor

localización. Estos datos de los que estamos hablando es el mapa de valores de construcción.

Este mapa, como su propio nombre indica, representa los valores de construcción del terreno, cuanto más alto sea el valor, mejor es para construir (este valor viene dado por el algoritmo de mapeado que se verá en el próximo apartado).

El campo de batalla es dividido en un tablero de 100 casillas de lado y el valor de cada una de estas casillas es almacenado en este mapa. Los distintos valores que estas casillas pueden tener son los siguientes:

- 0: valor normal.
- > 0 : cuando mayor es el valor mejor es para construir.
- < 0 : no se puede construir en estos valores, pero tienen datos importantes como: si es una rampa, un cambio de altura cerca de la rampa o un cambio de altura normal.

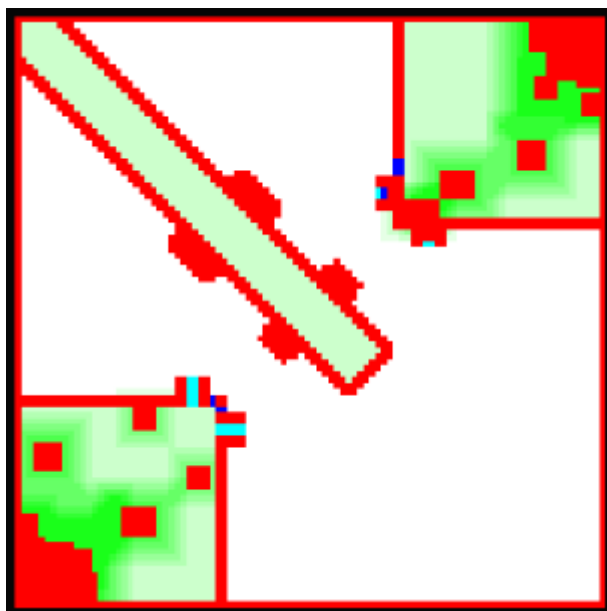


Ilustración 12: Captura del mapa de valores de construcción, donde se aprecian los distintos tipos de terrenos.

Realizar estos cálculos es muy costoso, así que se hace una sola vez cada vez que se cambia el mapa y se guardan los datos en un archivo binario, de esta forma la próxima vez únicamente hay que leerlo (cosa que es muchísimo más óptima).

Tanto los edificios como los parches de mineral actualizan el mapa de valores donde han sido creados para impedir que tanto donde ellos están como en las casillas de alrededor se pueda construir. Esto permite que haya espacio para que pasen las unidades. A parte de eso, los edificios tienen un radio de acción al construirse que lo que hace es mejorar el valor del terreno alrededor suyo a las casillas adyacentes se les suma un valor, a las adyacentes a éstas un valor menor, etc., creando así un cuadrado a su alrededor donde se ha aumentado de más a menos el valor del terreno. Esto consigue que en el algoritmo básico de construcción de los edificios, éstos tiendan a estar unos pegados con otros.

8.7.5. Algoritmo de mapeado

El mapa contiene gran cantidad de datos que, para nosotros, las personas, es muy fácil obtener. Cuáles son las mejores posiciones defensivas o donde es mejor construir determinado edificio son preguntas que para una persona es muy fácil responder con un simple vistazo. En cambio, la IA no es capaz de mirar un mapa y saber las respuestas a estas preguntas, necesita datos, necesita que este mapa se transforme en un sistema de datos que sea capaz de leer. En esto consiste este algoritmo, transforma el campo de batalla en un conjunto de datos que la IA es capaz de leer y entender.

Del campo de batalla, los datos que realmente obtenemos son los relacionados con las alturas del terreno: donde hay una rampa, donde hay acantilados desde donde puedo atacar a mis enemigos pero ellos a mí no, etc. Así que hemos de procesar el mapa para obtener dichos datos, y estos son los pasos que sigue el algoritmo.

Se divide el mapa en una cuadrícula de 100x100 y sobre cada cuadrícula se aplican los siguientes pasos: Se lanzan 4 rayos sobre cada una de las esquinas de esta casilla desde arriba y hacia abajo, según la altura en la que colisionan los rayos con el terreno nos da distintas informaciones. Si los cuatro rayos colisionan a la misma altura es que es una posición plana, cuanto más altura tiene mejor valor de construcción tendrá el terreno. Si colisionan en grupos de dos alturas diferentes pueden darse dos casos, el primero que esas alturas sean parecidas, en cuyo caso nos encontramos en una rampa, y si se diferencian por más altura es un acantilado. Cualquier otra combinación de rayos indica que es terreno abrupto y por tanto hace no se pueda construir ahí.

Con esta primera pasada sabemos dónde están las rampas, los acantilados y el terreno donde no se puede construir (los bordes del mapa). Pero como los acantilados únicamente nos interesan los que están cerca de rampas, ya que es donde se construirán las torretas defensivas (ver apartado ¿Dónde construir?) tenemos que descartar los acantilados lejanos de las rampas sustituyéndolos por terreno donde no se puede construir. En la imagen se observan los distintos colores:

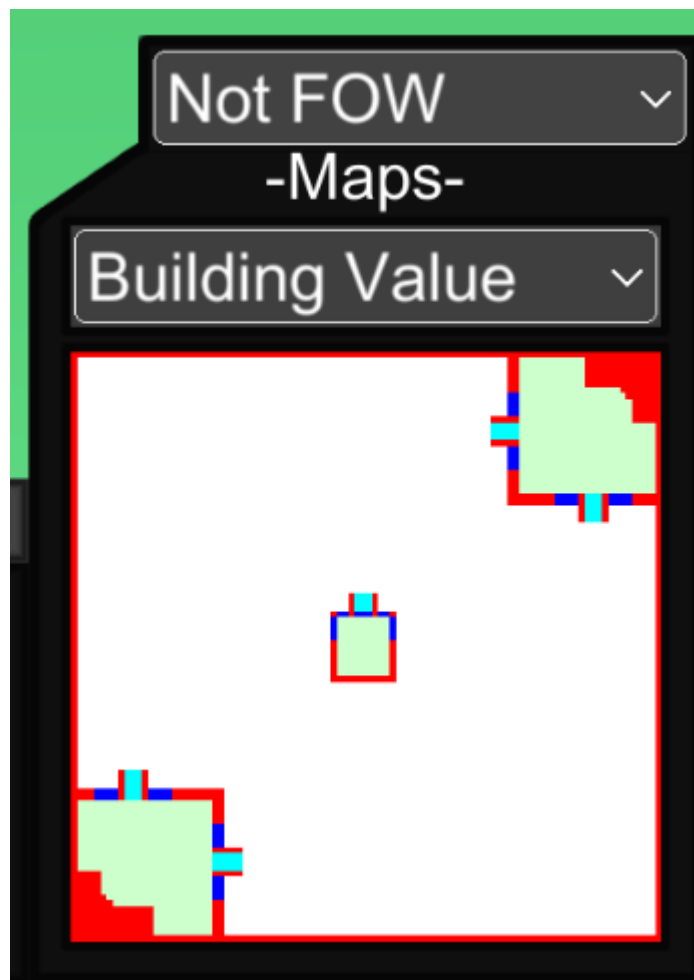


Ilustración 13: Mapa de valores de construcción donde se observan los distintos colores que representan distintos valores.

- Blanco: terreno de construcción normal.
- Verde: cuando más verde mejor es el terreno de construcción.
- Rojo: no se puede construir en ese terreno.
- Cian: Rampas

- Azul: Acantilados cercanos a las rampas, se usarán para la colocación de las torretas.

8.7.6. Distribución de gastos

Los recursos son un elemento limitado deben gastarse tanto en materia de economía (edificios y trabajadores) como en materia de combate (unidades de combate). Para ello, se establece un porcentaje a principio de partida que será la cantidad de recursos que se gastará en cada materia. Al principio de la partida nos interesa que el porcentaje de economía sea superior para poder desarrollarnos mejor, conforme avance la partida el porcentaje ira inclinándose hacia el combate, ya que la economía ya estará desarrollada.

Para saber qué porcentaje actual estamos destinando a cada materia, los gastos que se hacen tanto en economía como en combate se van guardando en unos contadores, dichos gastos se utilizan para calcular el porcentaje actual. Pero aquí surge otro problema, llegado un punto avanzado de la partida estos contadores son tan altos que los gastos que se hacen no cambian casi el porcentaje actual, por ejemplo, si tenemos 5000 en cada uno de los contadores, el porcentaje actual será de un 50% para cada materia, si el porcentaje deseado es 40 economía y 60 combate habría que destinar una gran cantidad de recursos a equilibrar ese porcentaje. Para solucionar esta situación y darles más peso a los gastos actuales que a los de hace tiempo, cada 30 segundos los contadores de gastos se dividen entre 2, esto hace que el porcentaje no varíe, pero hace que los gastos actuales tengan más repercusión sobre ellos.

8.7.7. Perfiles

Los perfiles son clases que guardan las variables que hacen a una IA única. Para obtener estas variables, se leen archivos externos con formato “.userData”. *Parseando* estos archivos obtenemos toda la información necesaria que luego será leída por los distintos *managers*.

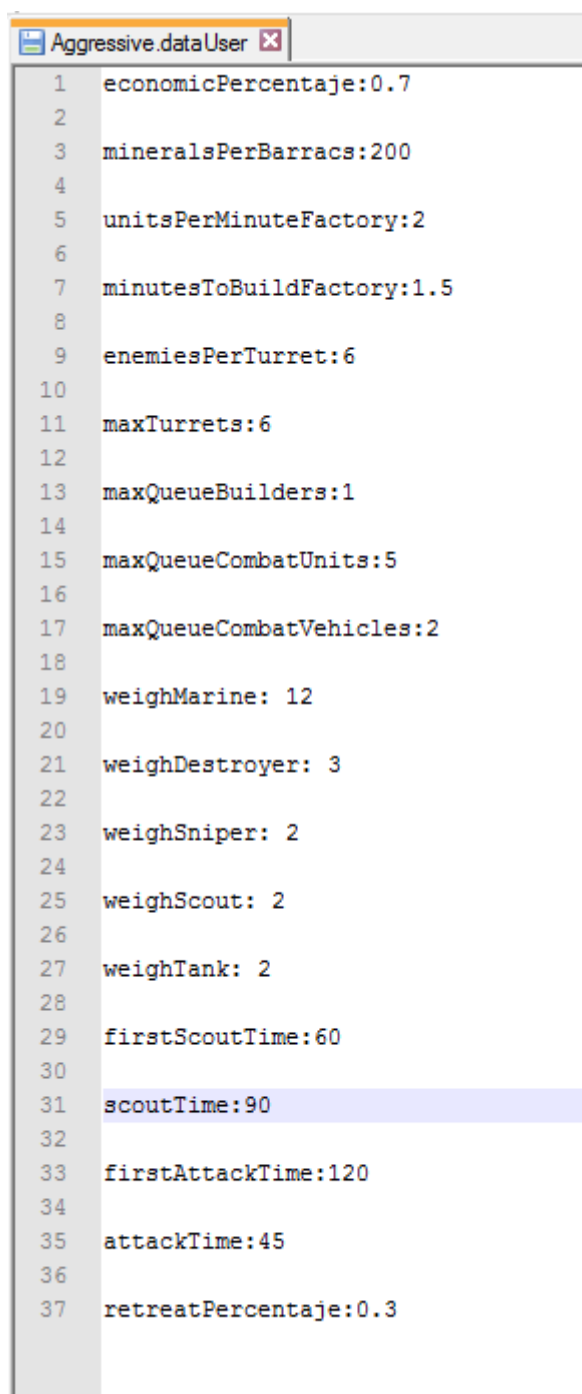
Estos perfiles nos permiten modificar el comportamiento de la IA sin necesidad de editar el código. Actualmente guardan los siguientes datos (fácilmente ampliable):

- Porcentaje de gasto en economía.
- Cantidad de minerales por minuto por *Barracs*.
- Unidades por minuto que queremos que se creen en una *Factory*.
- Cantidad de tiempo que queremos que la *Factory* tarde en hacer las unidades.
- Cantidad de enemigos por torreta.
- Cantidad máxima de torretas.
- Cantidad de unidades en la cola de producción.
- Ponderaciones de las unidades en las escuadras.
- Tiempos de exploración.
- Tiempos de ataque.
- Porcentaje para retirada en el ataque.

Todas estas variables, permiten cambiar el comportamiento de la IA. Editando estas variables podemos conseguir todo tipo de perfiles, desde más agresivos a más defensivos.

Los diferentes valores en estos perfiles, consiguen que la IA tenga diferentes respuestas en determinadas ocasiones. De esta forma podemos tener infinidad de IAs que se comporten de forma distinta únicamente editando un archivo de texto. Por ejemplo, editando los tiempos tanto de exploración como de ataque podremos hacer que una IA ataque antes que otra. O editando el valor de cantidad de enemigos por torreta se harán más o menos torretas. El cambio es estos valores supone un cambio en la reacción en diversas situaciones.

Podemos definir el conjunto de esas variables como la personalidad de la inteligencia artificial, estas variables serán lo único que diferencian una IA de otra.



```
1 economicPercentage:0.7
2
3 mineralsPerBarracs:200
4
5 unitsPerMinuteFactory:2
6
7 minutesToBuildFactory:1.5
8
9 enemiesPerTurret:6
10
11 maxTurrets:6
12
13 maxQueueBuilders:1
14
15 maxQueueCombatUnits:5
16
17 maxQueueCombatVehicles:2
18
19 weighMarine: 12
20
21 weighDestroyer: 3
22
23 weighSniper: 2
24
25 weighScout: 2
26
27 weighTank: 2
28
29 firstScoutTime:60
30
31 scoutTime:90
32
33 firstAttackTime:120
34
35 attackTime:45
36
37 retreatPercentage:0.3
```

Ilustración 14: Ejemplo de perfil, archivo Aggressive.dataUser

8.7.8. Escuadras

Todas las unidades de combate, a excepción de la unidad encargada de explorar, se agrupan en escuadras. Estas escuadras son a las que la IA da órdenes, y esta a su vez se las da a las unidades.

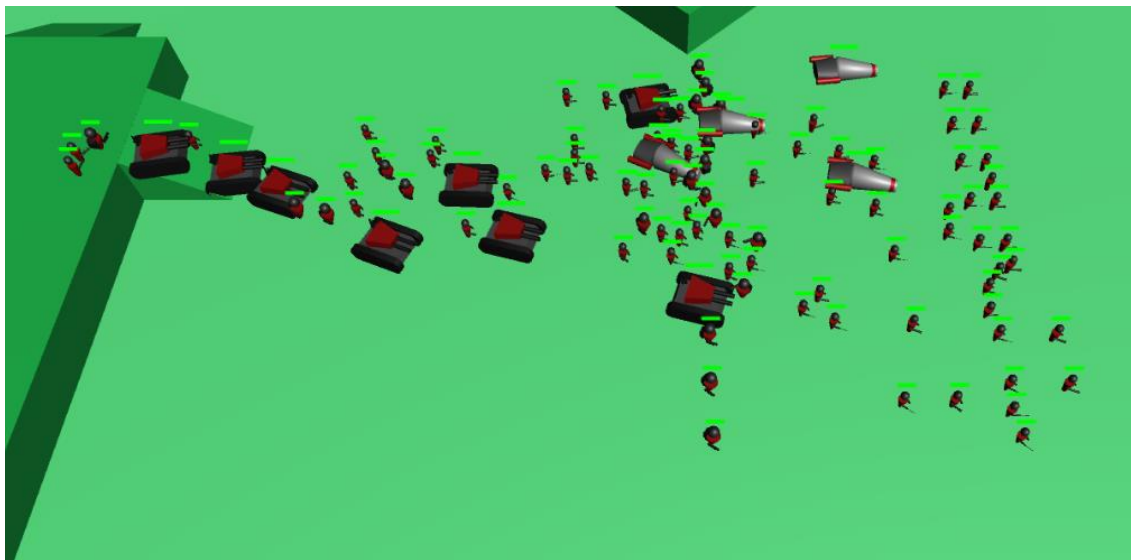


Ilustración 15: Escuadra poniéndose en posición

La IA tiene varias órdenes disponibles para darle a la escuadra: moverse a una posición, atacar a una posición o retirarse a base. Estas tres sencillas órdenes a simple vista, hacen los cálculos necesarios para que, a su vez, decirles a las unidades que conforman la escuadra que tienen que hacer. La escuadra de forma interna reordena y mueve las unidades, todo esto sin “molestar” a la IA principal. Se ha creado una especie de jerarquía donde todos los elementos tienen inteligencia propia y pueden tomar sus propias decisiones, pero estas supeditadas al eslabón superior.

Una escuadra puede estar creada por todo tipo de unidades, desde unidades de a pie hasta vehículos. La escuadra siempre va en formación. Mediante una lista de los tipos de unidades se dicta el orden en el que formarán las unidades. Los vehículos, dado su tamaño, siempre irán en distintas filas que las unidades de a pie. El número de filas y columnas de la formación dependerá del número de unidades, siendo el número de filas una cuarta parte del de columnas. Dado que los vehículos son mucho más grandes que las unidades de a pie, no puede haber el mismo número de ellas por fila. Así que se hace un cálculo para que el tamaño de estas filas sea igual, reduciendo el número de vehículos por fila.

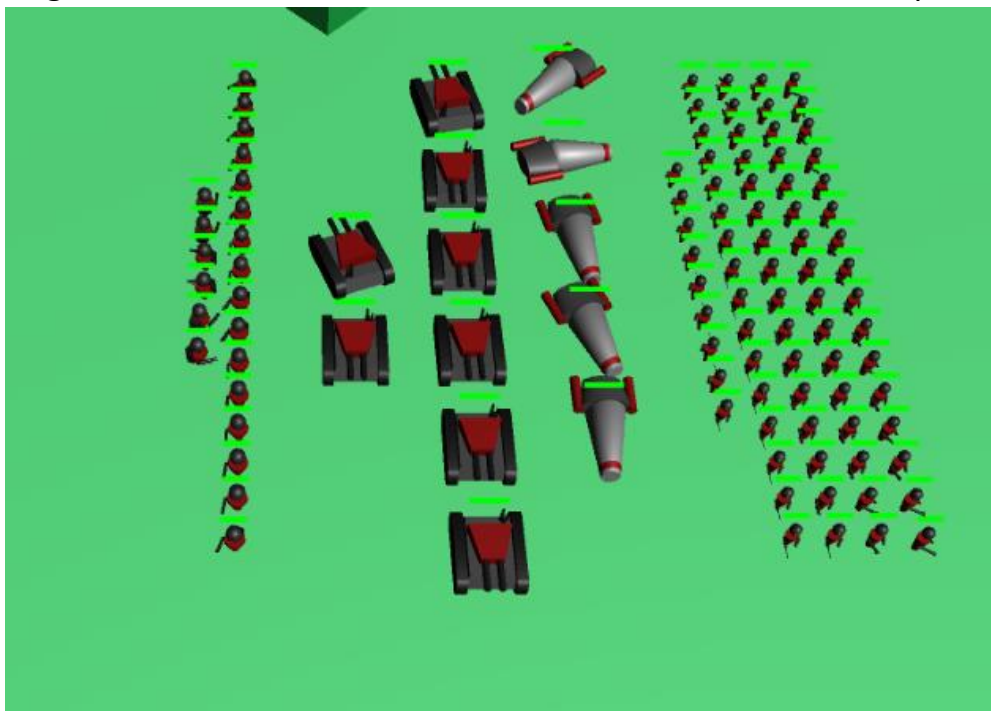


Ilustración 16: Escuadra ordenada con todo tipo de unidades.

La escuadra recalcula su tamaño y orden de forma automática cuando se añaden unidades a ella. Ocurre lo mismo cuando se pierden unidades. De forma interna, la escuadra tiene dos listas de la misma longitud, una de unidades, y otra de posiciones, a cada una de estas unidades le corresponde una posición, a la cual habrán de moverse si así se les ordena.

8.7.9. Parser

En el proyecto se leen varios archivos externos y se extraen variables de los mismos. De este proceso, llamado *parsear*, se encarga el parser. En este trabajo se han utilizado dos tipos de archivos, los archivos de texto, que son los que se quiere que se editen, y los archivos binarios, que no son editables. En los archivos de texto tenemos los perfiles, los edificios y unidades. Por otro lado, en los archivos binarios, tenemos los valores del mapa de construcción.

El proceso de *parseado* es diferente si es archivo de texto o binario. Para los de texto se guarda todo en una variable de la cual se va leyendo línea a línea, si esta línea contiene alguna variable esta se guarda. Las variables vienen definidas por el formato: "variable:valor". Luego con estas variables creamos los elementos que sean necesarios, elementos a los que la IA tendrá acceso. Por otro lado, *parsear* el archivo binario es más sencillo ya que es un proceso automático, existe un método de Unity que dándole el archivo devuelve los datos que hay en el interior, a estos datos tendrá acceso la IA.

8.8. Inteligencia Artificial

Como ya se ha mencionado, el desarrollo de la IA se planteó mediante una serie de preguntas que debían ir respondiéndose e implementándose. A continuación, se expondrán esas preguntas, las respuestas que se obtuvieron y como se implementó todo.

8.8.1. ¿Qué construir?

Una vez desarrollado el grueso del juego, se empezó a programar la IA respondiendo a esta pregunta. Que edificio construir en cada momento depende de una serie de funciones que lo deciden. Esta serie de funciones,

una para cada tipo de edificio, se ejecutan una vez por segundo. Si se cumplen las condiciones que se plantean en estas funciones, se añade la construcción del edificio a la cola de tareas. Las condiciones para crear cada edificio son las siguientes:

Supply: si quedan menos de 3 suministros para alcanzar el límite de suministros actual, o si quedan menos que la mitad de edificios de producción de unidades (ej. Si tengo 10 barracas y quedan menos de 5 para bloquearse). Estos valores pueden ser editados en el perfil para adaptarse al tipo de IA que se desee tener.



Ilustración 17: Supply

Barracs: la cantidad de minerales por minuto que se producen se divide entre un número determinado por el perfil, ese resultado es el número de *Barracs* que se tienen que tener. Normalmente, ese número predeterminado será más o menos la cantidad de recursos que puede

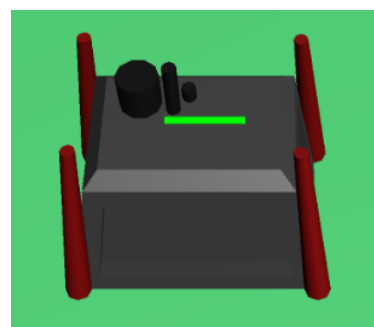


Ilustración 18: Barracs

gastar una barraca en un minuto creando unidades. De esta forma, se obtiene el número óptimo de barracas, pero este número puede cambiar dependiendo si queremos un perfil más agresivo, o defensivo.

Factory: se cuentan la cantidad de unidades que se han de construir en *Factories*. Tenemos dos datos, las unidades que queremos que produzca cada *Factory* en un minuto y el tiempo que queremos que se tarde en hacer todas las unidades, dependiendo de esos datos y de la



Ilustración 19: Factory

cantidad de unidades a construir se calcula el número de *Factories* que hay que tener.

Turrets: se cuentan la cantidad de unidades enemigas que se conocen y se divide entre un número dado por el perfil seleccionado. Esta es la cantidad de torretas que se tienen que construir, pero hay un número máximo que también está determinado por el perfil. Si no se está construyendo ninguna torreta ni hay tareas para ello, se creará una torreta siempre que no se supere el máximo anteriormente calculado.



Ilustración 20: Turret

Academy: la academia solo se puede construir si alguna otra tarea lo pide.



Ilustración 21: Academy

Main Building: no se puede construir un segundo *Main Building*.

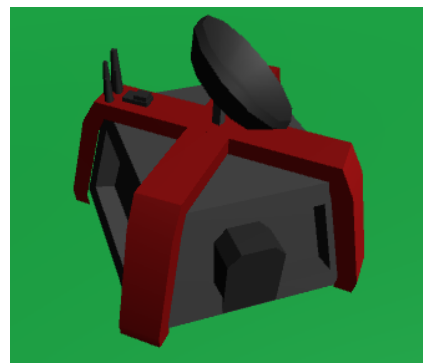


Ilustración 22: Main Building

Si se cumplen las condiciones de construcción, no se construirá el edificio directamente, sino que se añadirá la tarea de construcción del mismo. Dicha tarea tendrá más o menos prioridad dependiendo del edificio. Por lo tanto, el edificio se construirá en el momento que la tarea sea la primera de la lista y se tengan materias suficientes para ello. Lo que hay que destacar, es que mientras un edificio está en construcción o en la lista de tareas, no se crean más tareas con el mismo propósito, es decir, no puede haber dos tareas de construir *Supply*. Hasta que una no se complete no se crea otra.

8.8.2. ¿Dónde construir?

La gran mayoría de edificios se construirán según el algoritmo básico de construcción. Pero habrá edificios que necesiten otro tipo de algoritmos. Ahora mismo el único edificio que no utilizará el algoritmo básico es la torreta. Esta se colocará en la mejor posición defensiva.

8.8.2.1. Algoritmo de construcción básico

Bajo la premisa de que todos los edificios tienen que construirse unos cerca de otros, pero con espacio para dejar las unidades pasar y que cada partida sea diferente. El algoritmo tiene acceso a todos los datos del mapa de valores de construcción, de esta forma cuando se hable de casillas o posiciones, todos los datos los sacará de este mapa.

El algoritmo devuelve un lugar para construir el edificio, y como queremos que ese lugar sea cada vez uno distinto se utilizan en determinados puntos números aleatorios, esto provoca que pueda fallar al elegir un lugar ya ocupado, para solucionar esto cada nuevo intento, si el anterior falla, se va aumentando un contador que hará más fácil la elección de un hueco vacío, pero a su vez hará que este más lejos del resto de los

edificios, así que primero se intenta con el contador con un número bajo y poco a poco va subiendo si es necesario.

La primera parte del algoritmo es obtener el centro de los edificios. Mediante las posiciones de nuestros edificios, sabemos dónde está la media de todos ellos. Desde este centro se traza un anillo donde se buscará el mejor sitio de construir. Utilizamos un anillo porque las posiciones centrales ya estarán ocupadas por otros edificios, queremos ocupar las afueras de nuestra base. El radio interior de este anillo depende de dos cosas, de un número fijo que nosotros le damos, y del contador anteriormente mencionado, cuando mayor sea el contador mayor será el radio interior. Por otro lado, el radio exterior del anillo depende tanto de un número fijo, como del contador y también del número de edificios que tengamos (si tenemos muchos edificios no queremos que el radio exterior sea muy pequeño y no encuentre ningún hueco vacío).

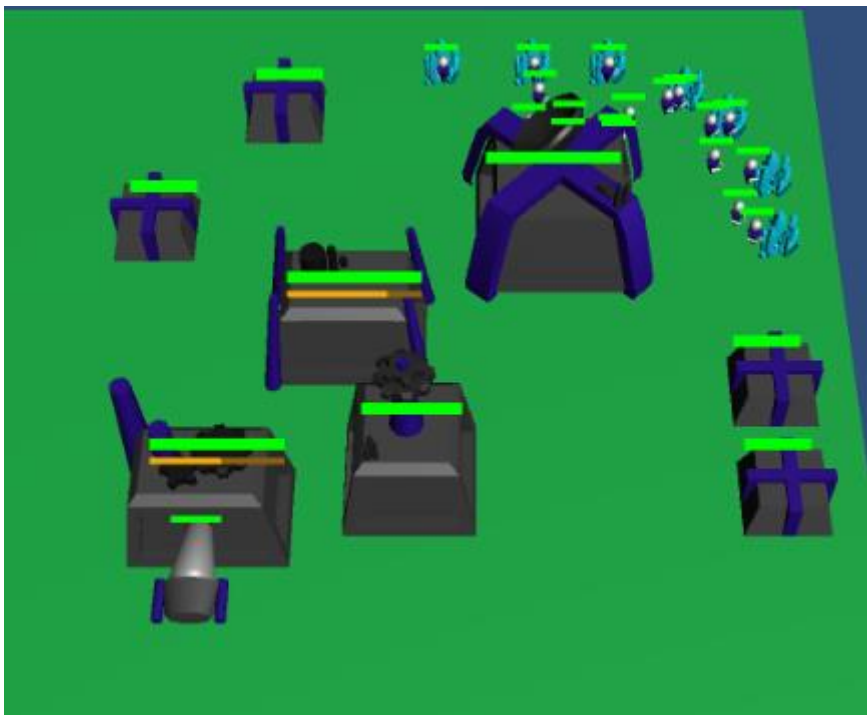


Ilustración 23: Edificios colocados mediante el algoritmo de construcción.

De este anillo alrededor del centro de los edificios se escoge un punto aleatorio, y desde este punto se traza un círculo donde seleccionamos todas las casillas alrededor y aquí es donde realmente se utilizan los valores del mapa de valores de construcción. De estas casillas seleccionadas se escogen las que más valor tenga y de todas ellas una aleatoria. Se comprueba si en esta casilla se puede construir el edificio, es decir que cabe el edificio sin que este ocupe ninguna casilla donde no se puede construir. Si se puede, se construye el edificio, por el contrario, si falla se descarta esa casilla y se prueba otra de las que tenían el valor más alto. Si aun así todo esto falla el algoritmo se reinicia y aumenta en uno su contador haciéndolo más fácil que acierte.

Existe una barrera de seguridad, si el contador alcanza 200 se cancela la construcción del edificio, porque se determina que no se puede construir.

8.8.2.2. Algoritmo de construcción de torretas

Las torretas queremos construirlas cerca de las rampas y en terreno elevado para que tengan ventaja sobre los atacantes. Sabiendo esto se creó el siguiente algoritmo para colocar las torretas.

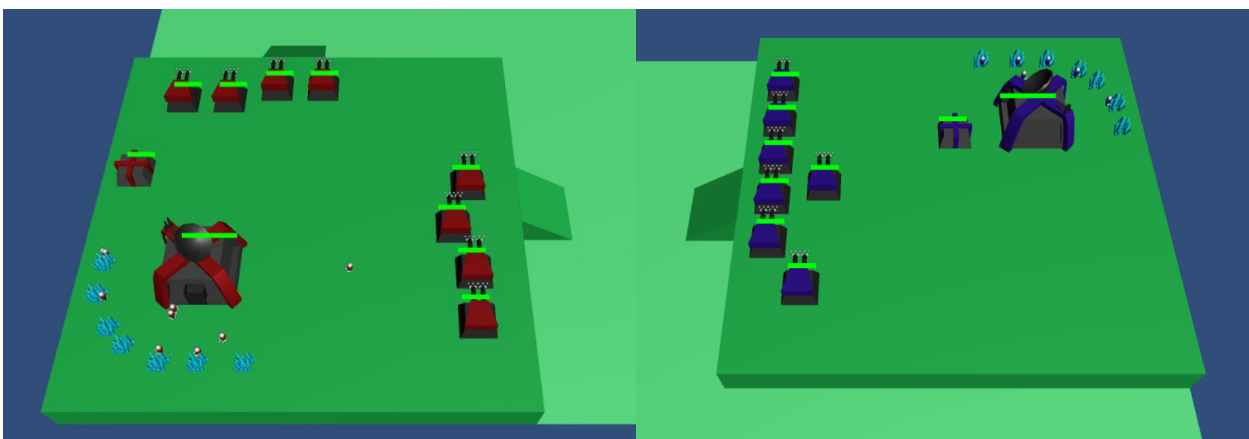


Ilustración 24: El jugador Rojo construye sus torretas en los dos lados del acantilado porque tiene dos rampas. En cambio, el Azul solo en uno solo, porque en el otro lado no hay rampa que defender.

Se obtiene una lista de todos los acantilados que hay cerca de rampas del mapa de valores de construcción. De estas posiciones se descartan las que están lejos de nuestra base, ya que esto haría que pudiéramos construir torretas en la rampa del enemigo. Una vez hecha la criba, escogemos una de esas posiciones de forma aleatoria. Desde esta trazamos un círculo de casillas, se escoge una de esas casillas al azar, si esta casilla está en la parte alta del acantilado y cabe la torreta, esta se construye, sino se prueba con otra posición. Si en este círculo ninguna posición es buena se escoge otro punto de la lista de acantilados cercanos a rampas.

8.8.3. ¿Qué unidades hacer?

La cantidad de unidades que se hará de cada clase dependerá en todo momento de un porcentaje. Dicho porcentaje cambiara a lo largo de la partida, en función de la cantidad y porcentaje de las unidades del enemigo ya descubiertas.

El cálculo para obtener el porcentaje inicial es el siguiente, se tienen unas ponderaciones de forma predeterminada (por ejemplo, 10 *marines*, 2 *destroyers* y 1 *sniper*), a partir de estos números se sacan los porcentajes de forma que se cumplan dichas ponderaciones, cada 10 *marines* habrá 2 *destroyers* y un *sniper*, y así con el resto de las unidades. A la hora de decidir que unidad se hace se mira que unidades tienen su porcentaje actual por debajo del deseado y se crea ese tipo de unidad. Por ejemplo, tenemos 1 *marine*, 1 *destroyer* y 1 *sniper*, en cambio las ponderaciones son 10, 2 y 1 respectivamente, pues se crearán 9 *marines* y 1 *destroyer* antes de intentar crear otro *sniper*. Los números elegidos en este párrafo han sido un ejemplo, realmente, los números son dados por el perfil de la IA. Por tanto, tanto las ponderaciones y los porcentajes cambian de un perfil a otro.

El objetivo de que los porcentajes cambien según las unidades del enemigo es poder adaptarse a la combinación de unidades del enemigo lo mejor posible. Las ponderaciones únicamente sirven para el porcentaje inicial, el objetivo es que estas varíen a lo largo de la partida. Porque a la hora de atacar es como piedra, papel, tijera, hay ciertas unidades que son mejores contra unas y peores contra otras.

8.8.4. ¿Cuántas unidades hacer?

Los trabajadores son las únicas unidades que tienen un tope preestablecido. Este tope es la cantidad de trabajadores que pueden trabajar a la vez de forma óptima, un total de 3 por cada mena de mineral. Se calculan las menas de mineral cercanas y ese número, por tres, es la cantidad tope de trabajadores que se obtiene.

Por otro lado, el resto de unidades de combate no tienen un límite como tal. El único límite que tienen es la cantidad máxima de suministros, a partir de ese número no se podrán crear más unidades. Así pues, para las unidades de combate se crearán el número necesario para equilibrar los porcentajes y de ahí se empezarán a crear hasta el máximo de suministro manteniendo los porcentajes lo mejor posible.

8.8.5. ¿Cuándo hacer las unidades?

El *resource manager*, el encargado de decidir en qué y cuándo se gastan los recursos, sabe qué porcentaje se ha de gastar en cada materia. La gestión de recursos se hace de forma automática, así que los recursos destinados a hacer las unidades dependen de esta gestión. Cuando más recursos se obtengan y cuando más se avance en la partida, más recursos se destinarán a materias de combate y por tanto se harán las unidades.

Por otra parte, si se ve que el enemigo tiene más unidades que él los porcentajes cambiaran, permitiendo hacer más unidades. Normalmente, la producción de unidades está bloqueada a menos que se permita gastar recursos en materias de combate, pero en caso de ser necesario, esta restricción de puede pasar por alto para crear las unidades que se necesiten.

8.8.6. ¿Cómo explorar?

Explorar o scautear, tiene como objetivo saber qué unidades y edificios tiene el oponente en ese momento. Del perfil que carguemos obtenemos dos variables que usaremos aquí, una es el tiempo que se tarda en hacer la primera exploración y la otra el tiempo que se tarda entre exploraciones.

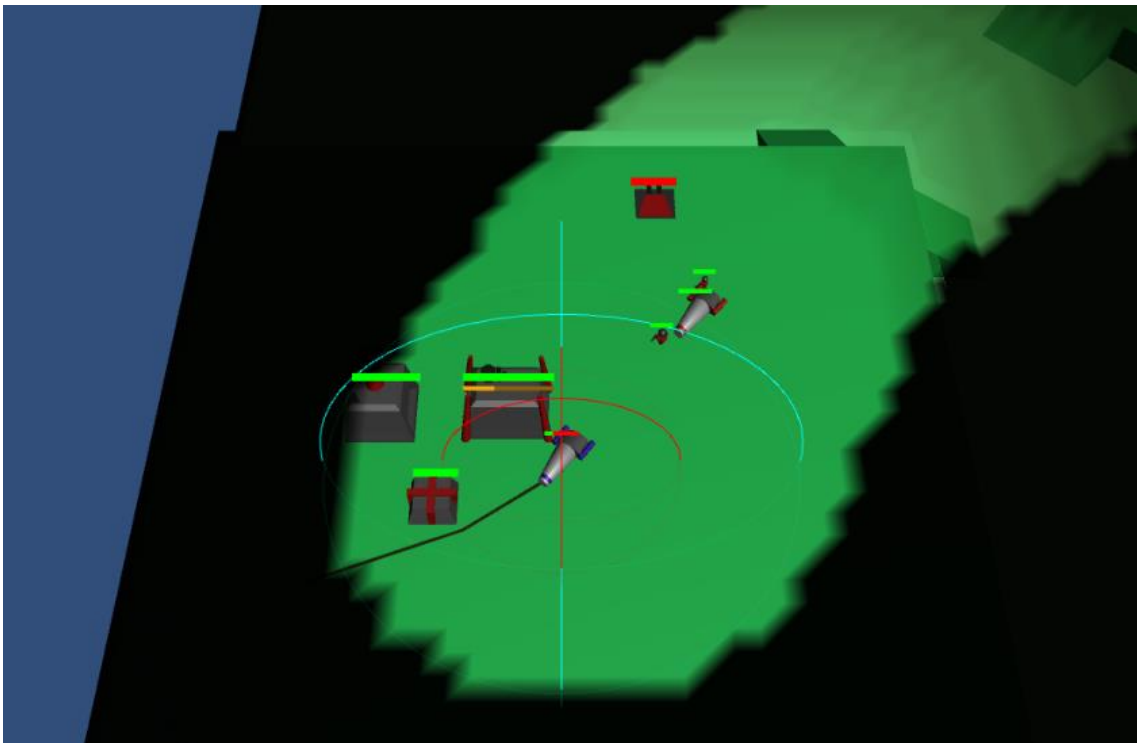


Ilustración 25: Explorador de la IA azul obteniendo datos de la IA roja.

El *combat manager* se encargará de crear una unidad “*scout*” que no formará parte de ninguna escuadra, tendrá como único objetivo explorar la base enemiga. Tras crear la unidad y haber pasado el tiempo de espera, la unidad irá a la base del oponente y volverá. Los datos que obtenga se usaran posteriormente para la fase de ataque. Tras volver de explorar la unidad esperará en la base hasta que pase el tiempo de espera y repita el proceso.

En caso de muerte del “*scout*” en el proceso de exploración el *manager* se encargará de crear otro para asignarle la tarea.

8.8.7. ¿Dónde atacar?

Hay dos mecánicas base de ataque. La primera es el ataque a la base enemiga y la segunda es el ataque a las unidades que nos están atacando a nosotros.

La primera es bien sencilla, dada la simetría del mapa, sabemos dónde se encuentra la base enemiga, así que sabremos donde tenemos que atacar.

La segunda es un poco más compleja. La escuadra está formada por varias unidades. Para cada de que una de ellas este siendo atacada, el resto de unidades de la escuadra que no estén en modo de ataque irán en su ayuda.

8.8.8. ¿Cuándo atacar?

Al igual que para explorar, tenemos dos variables que sacamos de los datos del perfil. Estas variables corresponden al tiempo inicial de atacar y el tiempo entre ataques. Como no queremos que estos tiempos sean siempre los mismos, al contador de tiempo le añadimos una cantidad de segundos aleatoria entre 0 y la mitad del tiempo (ej. Si el tiempo de ataque inicial es 60 el contador quedara en $60 + [0-30]$). Una vez que el tiempo de ataque llega a 0 se prepara para atacar, pero no lo hará si sabe que tiene menos unidades que el enemigo (dato que saca de explorar). En el momento que el contador este en 0 y tenga más unidades será el momento de atacar.

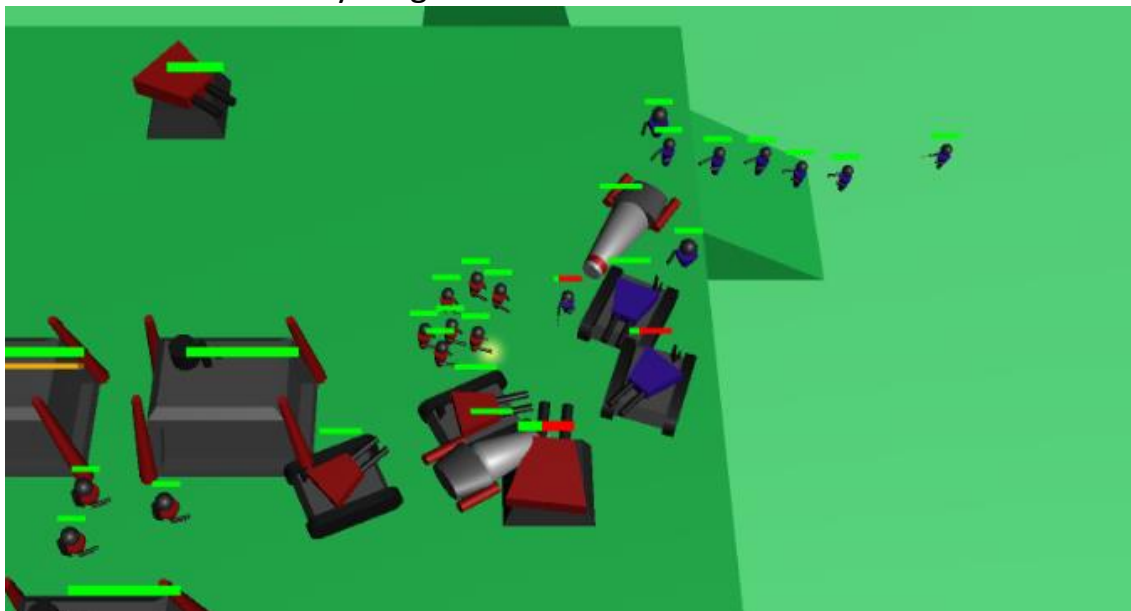


Ilustración 26: Batalla entre las dos facciones, la roja cuenta con la ventaja defensiva

8.8.9. ¿Con qué atacar?

Cada jugador tiene dos escuadras, una de ataque y una de defensa. Todas las unidades, cuando se crean, se unen a la escuadra de defensa. Cuando se decide atacar, todas las unidades de la escuadra de defensa pasan a las de ataque. Y la de ataque les dice de moverse a la base del enemigo.

Cuando estamos atacando la base del enemigo, si tenemos menos de un tanto por ciento de las tropas del enemigo, la escuadra se retira, el ataque ha fallado. Este tanto por ciento viene dado por el perfil, en un perfil más agresivo el porcentaje será más bajo, haciendo más difícil la retirada, en uno más defensivo, al contrario.

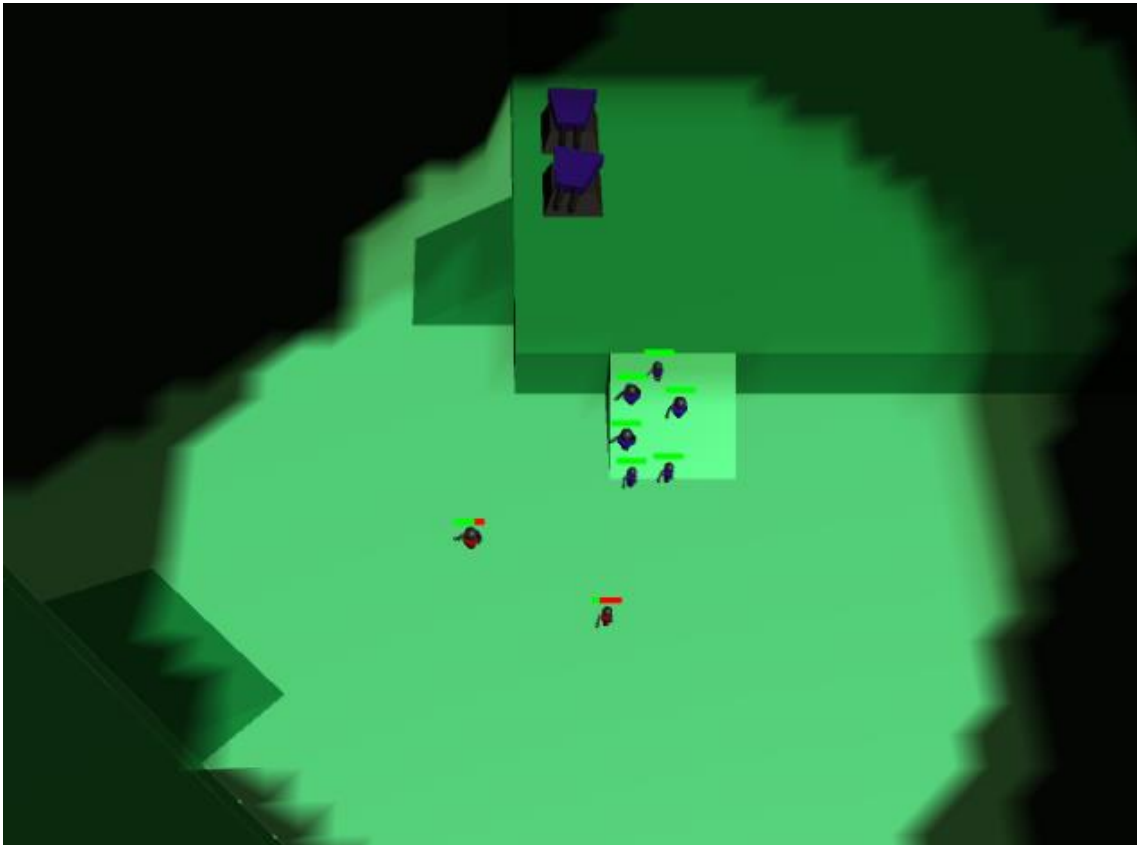


Ilustración 27: IA roja retirándose a base con las pocas fuerzas restantes tras un ataque fallido.

8.9. Sistema de depuración visual

Siendo la depuración visual de la partida uno de los principales objetivos del *TFG*. Se han creado distintas formas de mostrar de forma visual los datos de la partida, intentando que ninguna de estas sea muy intrusiva para poder observar de forma cómoda lo que está ocurriendo en cada momento en la partida.

8.9.1. Gizmos

Los gizmos son figuras o iconos que se muestran sobre los modelos 3D para dar información sobre ellos. Unity cuenta con gizmos automáticos que nos dicen donde hay una luz, un sistema de partículas o una cámara, pero estos no nos interesan a nosotros, ya que no dan datos relevantes sobre la partida.



Ilustración 28: Builder trazando el camino del Main Building al mineral.

En este caso, el sistema de Unity de gizmos se usa para mostrar los radios de visión y de ataque de la unidad o edificio seleccionado y para mostrar el camino que está siguiendo la unidad a la hora de moverse.

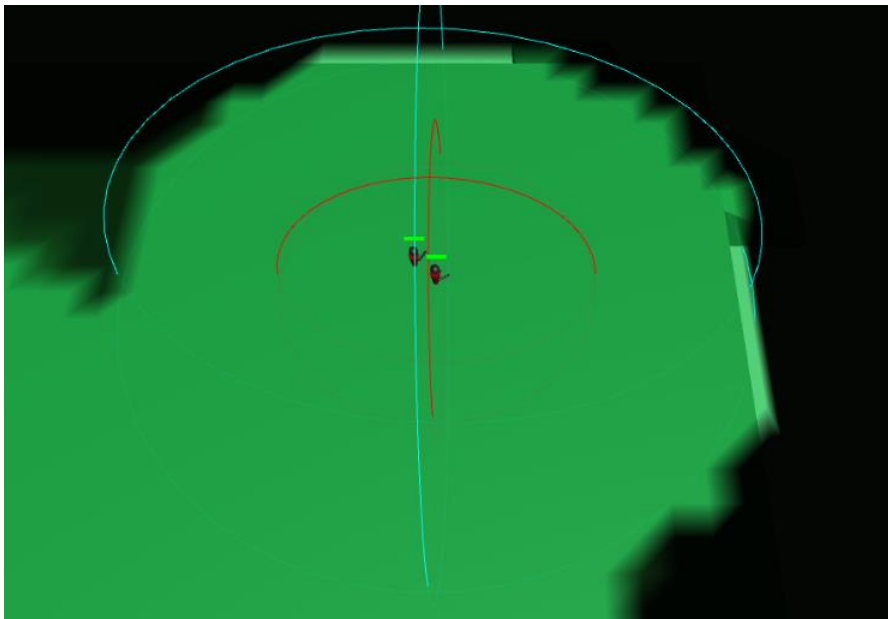


Ilustración 29: Gizmos del radio de visión (azul) y del radio de ataque (rojo). Encima de las unidades se pueden apreciar las barras de vida.

Se podrían mostrar gran cantidad de gizmos, pero dado que son la forma de depuración visual más intrusiva, ya que se ponen en los modelos 3D, se decidió no abusar de ellos.

En este apartado, pese a no ser gizmos como tal, vamos a añadir las barras de progreso que tienen las distintas entidades, ya que se muestran sobre los modelos. Hay dos tipos de barras: las de vida, que las tienen tanto unidades y edificios, y que como su nombre indica nos dicen la vida restante de la entidad, y las de progreso, que solo tienen los edificios y nos muestran el progreso en la creación de una unidad.

8.9.2. Niebla de guerra (FOW)

La niebla de guerra o Fog of War (*FOW*) es uno de los elementos más característicos de los *RTS* y de los juegos de estrategia en general. El objetivo de esta niebla es ocultar lo que un jugador no ha visto. Hay dos niveles de niebla de guerra:

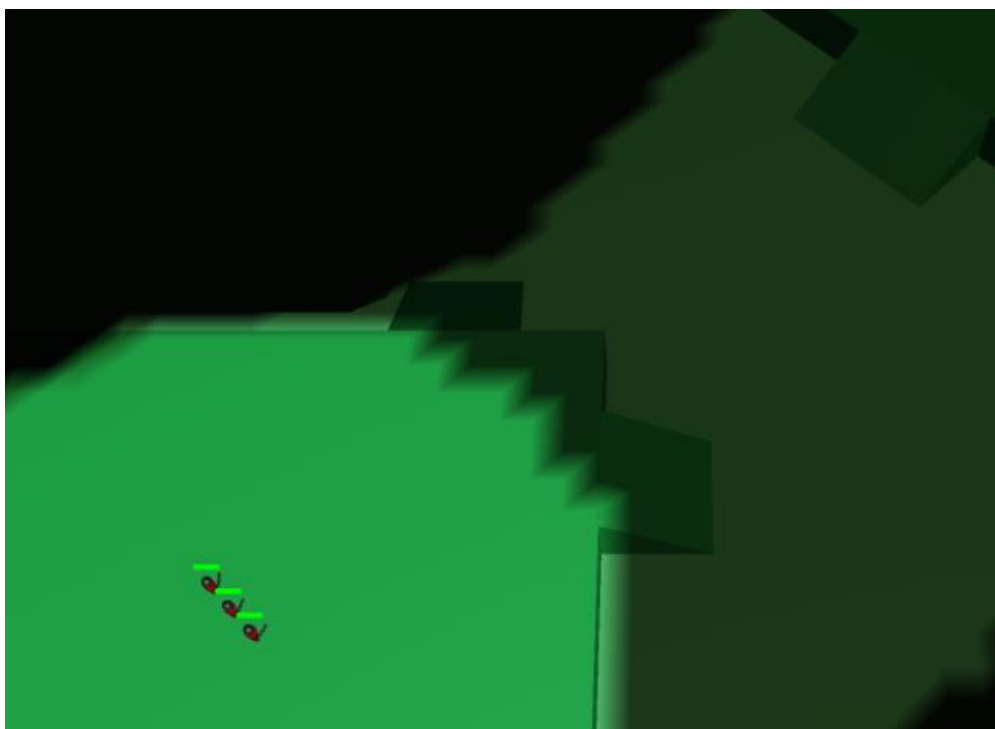


Ilustración 30: Zona del mapa donde se aprecian los tres estados de la niebla de guerra.

- Zonas inexploradas: este tipo de niebla puede ser o totalmente opaca (no mostrando ni el terreno que hay debajo) o parcialmente opaca, es la más oscura de las dos nieblas y tras ella esconde las zonas inexploradas del mapa. En este desarrollo en concreto es parcialmente opaca, y es la única niebla que oculta los edificios.
- Zonas exploradas, pero no visibles: este tipo de niebla se forma en las zonas del mapa que ya hemos explorado pero que no estamos viendo en este momento. Suele ser translúcida, oscureciendo un poco el terreno que tiene debajo. Esconde a las unidades que tiene debajo, pero no los edificios que ya han sido vistos (los que se han construido posteriormente sobre esta zona no son visibles hasta que se vuelve a explorar).

En este proyecto, el *FOW* se ha obtenido de un repositorio gratuito en internet, creado por **Fredrik Holmström**⁷, pero se ha editado para poder ajustarse a las necesidades que había. Se ha leído y entendido el código para poder editarlo, pero no se ha realizado desde cero dada la complejidad y la cantidad de trabajo que suponía (ya que no era una parte esencial, solamente visual).

8.9.3. Interfaz

La interfaz es el sistema de depuración más complejo y el que más información ofrece. Como se dice es una interfaz que se sobrepone a la pantalla y que da gran cantidad de datos para cada jugador. Se ha creado de forma que sea fácilmente ampliable para poder mostrar los datos que sean necesarios. Consta de 4 partes principales:

⁷ Aquí está el link al repositorio del autor: <https://github.com/fholm/unityassets>

8.9.3.1. Cuadros superiores

En ellos se muestra una pequeña cantidad de datos, pero son datos que interesa tener a la vista en todo momento. Se separa en 3 recuadros, uno central y uno a cada lado. El central muestra el jugador seleccionado, este jugador será del que se muestre toda la información en la interfaz. El izquierdo muestra la cantidad de minerales que el jugador tiene en cada momento al igual que la cantidad de minerales por minuto que está obteniendo. Por último, el de la derecha del todo muestra los suministros actuales y máximos que tiene el jugador.

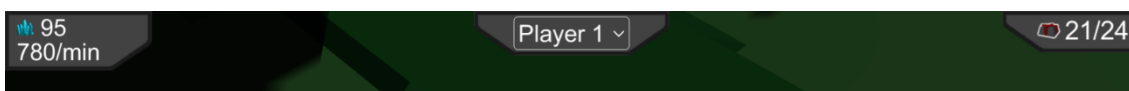


Ilustración 31: Interfaz – cuadros superiores.

8.9.3.2. Consola de mensajes

Ubicada en la parte inferior izquierda, muestra una serie de mensajes que se van mandando a lo largo de la partida. Determinados eventos generan mensajes que son recogidos por la consola y mostrados por pantalla. Hay dos tipos de mensajes: los mensajes de texto, que solo dan información y los mensajes de posición, que si los clicas te llevan a una posición del mapa.

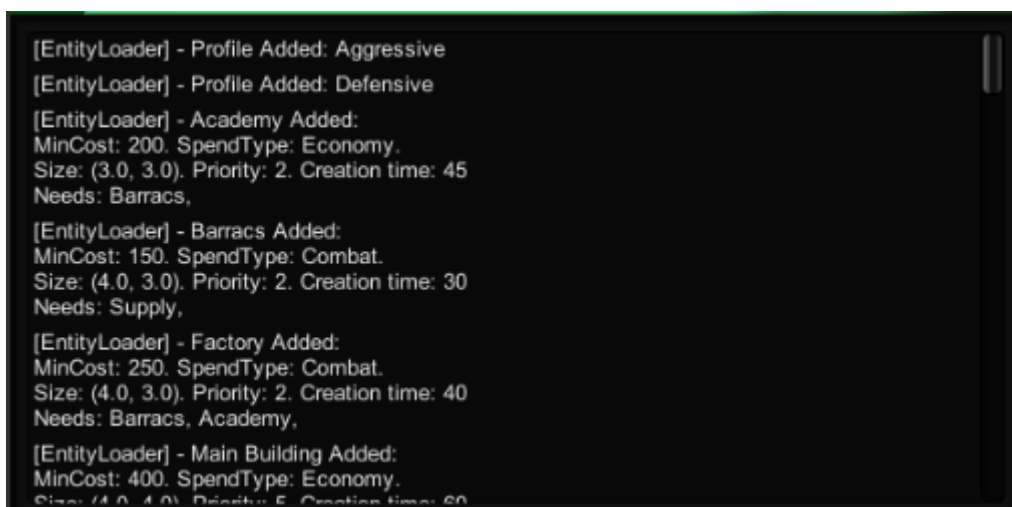


Ilustración 32: Interfaz - consola de mensajes.

Por ejemplo, cuando se crea un constructor, se crea un mensaje de texto que nos dice los suministros que tiene en ese momento ese jugador, y también se genera un segundo mensaje de posición que nos permite ir a la posición donde se ha creado dicho constructor.

8.9.3.3. Zona del minimapa

La zona del minimapa ubicada en la parte inferior derecha consta del minimapa propiamente dicho y de dos desplegables.



Ilustración 33: Interfaz - zona del minimapa.

El desplegable inferior nos permite seleccionar el tipo de mapa que queremos que se muestre en el minimapa. En el proyecto hay dos tipos de minimapa, el minimapa normal, que muestra lo que está sucediendo en el mapa mediante una vista cenital, y el otro mapa es el de valores de

construcción que mediante distintos colores nos dice que tipo de terreno es.

El desplegable superior, nos permite seleccionar que tipo de *FOW* queremos tener, tanto en el mapa como en el minimapa. Hay cuatro opciones: visión de la *IA 1*, visión de la *IA 2*, visión de las dos *IAs* y *FOW* desactivado.

8.9.3.4. Cuadro de información

Se encuentra en la parte inferior central de la pantalla. Es el elemento que muestra la información más importante. Se divide en cinco pestañas y un cuadro donde se muestra la información.

La primera de estas pestañas llamada "Info Selection" da información sobre la unidad o edificio seleccionado, su nombre, vida, ataque, radio de visión, estado, etc. Además de esto consta de un recuadro a la izquierda que es una cámara siguiendo a la unidad seleccionada en cada momento. Por último, en la parte superior derecha del recuadro, hay un botón con el texto "follow" (seguir en inglés) que si es clicado la cámara principal comenzara a seguir a la entidad seleccionada hasta que movamos la cámara, sigamos a otra entidad o esta muera.

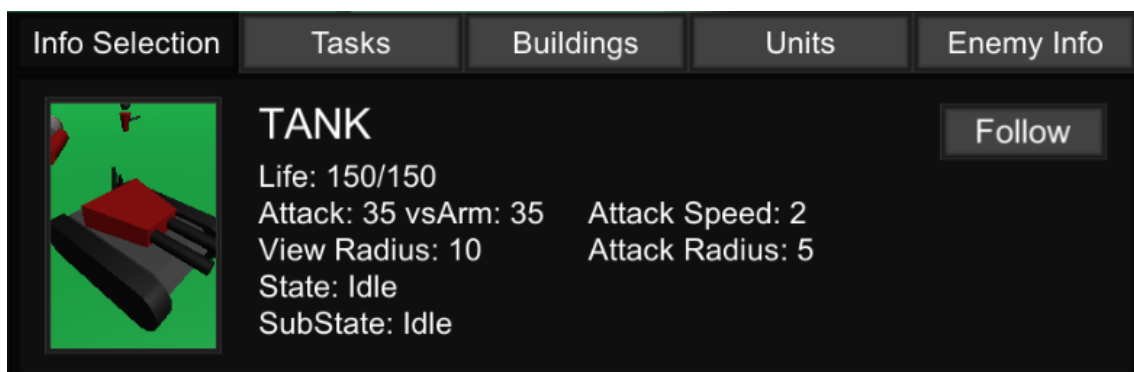
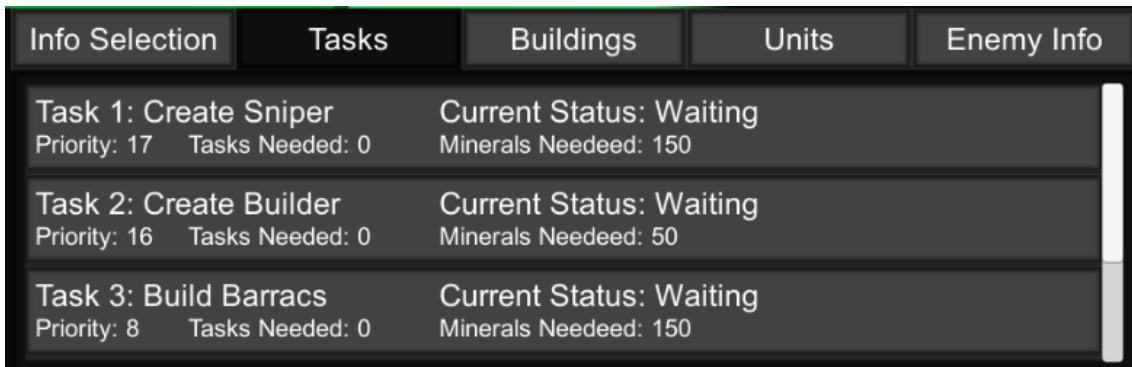


Ilustración 34: Interfaz - cuadro de información, pestaña "Info Selection".

La segunda pestaña, “*Tasks*”, muestra una lista de tareas pendientes del jugador seleccionado ya ordenadas. De dichas tareas sabremos: que tarea es (ej. Construir Marine), el estado actual, la prioridad, el número de tareas pendientes para poder ejecutarse, y los minerales necesarios para realizarla.



Info Selection	Tasks	Buildings	Units	Enemy Info
Task 1: Create Sniper		Current Status: Waiting		
Priority: 17	Tasks Needed: 0	Minerals Needed: 150		
Task 2: Create Builder		Current Status: Waiting		
Priority: 16	Tasks Needed: 0	Minerals Needed: 50		
Task 3: Build Barracs		Current Status: Waiting		
Priority: 8	Tasks Needed: 0	Minerals Needed: 150		

Ilustración 35: Interfaz - cuadro de información, pestaña "Tasks".

En la pestaña “*Buildings*”, tenemos una lista de los edificios que tiene el jugador seleccionado. De estos nos mostrara varios datos, como el nombre, tamaño, vida, o estado del edificio. Por último, tendremos un botón con el texto “Goto” por cada uno de los edificios que nos permitirá situar la cámara en dicho edificio y seleccionándolo para poder ver sus datos en la pestaña de “Info Selection”.



Info Selection	Tasks	Buildings	Units	Enemy Info
Size: 4x4		State: Creating Unit		
Supply	Life: 100/100		Go to	
Size: 2x2	State: Idle			
Barracs	Life: 100/100		Go to	
Size: 4x3	State: Creating Unit			
Academy	Life: 100/100		Go to	
Size: 2x2	State: Idle			

Ilustración 36: Interfaz - cuadro de información, pestaña "Buildings".

La de “Units”, es muy parecida a la anterior, pero este caso tiene información sobre las unidades del jugador seleccionado. Esta información consiste en, el nombre, la vida y el estado. Además del botón “Follow” que al clicarlo hará que la cámara siga a dicha unidad y seleccionándola.



Ilustración 37: Interfaz - cuadro de informacion, pestaña "Units".

La última de todas, la de “Enemy Info”, nos dice que información tiene cada IA sobre su contrincante. Es una lista de la cantidad de unidades y edificios que sabe que tiene.

Info Selection	Tasks	Buildings	Units	Enemy Info
Buildings		Units:		
- Main Buildings:	0	- Builders:	1	
- Barracs:	0	- Marines:	3	
- Academics:	0	- Destroyers:	1	
- Supplies:	0	- Snipers:	0	
- Factories:	0	- Tanks:	0	
- Turrets:	2	- Scouts:	1	

Ilustración 38: Interfaz - cuadro de informacion, pestaña "Enemy Info".

9. CONCLUSIONES

La valoración general del trabajo ha sido muy buena. Se han cumplido los objetivos y obtenido gran cantidad de conocimientos en el proceso. Se ha indagado sobre la historia de los RTS, se han estudiado sobre las mecánicas de un RTS y aplicado al proyecto, se ha creado una inteligencia artificial para un juego de estrategia capaz de completar una partida contra otra de su mismo tipo, informando mediante una interfaz al usuario que observa la partida.

Se ha aprendido que tanto la realización de un RTS convencional como la de una inteligencia artificial para este es un proceso muchísimo más complejo de lo que podría parecer. Se puede ampliar con decenas de mecánicas y comportamientos distintos. Parece que siempre va a haber cosas que mejorar en la inteligencia artificial, mecánicas que programar o unidades que añadir. Pese a todo el resultado es mucho mejor de lo esperado.

Gracias a la complejidad del proyecto se han tenido que mejorar tanto la forma de programar, como de organizarse. Resolviendo complejos problemas que se han ido encontrando por el camino. Se ha creado una inteligencia capaz de observar y obtener información de su entorno, procesarla y adaptarse lo mejor posible a la situación.

10.GLOSARIO

10.1. Definiciones

- **Barracs** (barracas): tipo de edificio empleado para crear las unidades de a pie (Marine, Destroyer y Sniper).
- **Building Manager**: Manager encargado de la creación y gestión de los edificios.
- **Civilization Manager**: Manager general, no tiene un propósito propio más que permitir comunicarse al resto de managers entre ellos, actuando como nexo de unión. También se encarga de cargar el perfil de la IA.
- **Combat Manager**: Manager encargado del combate, tiene acceso a las unidades del jugador, a los edificios, a las escuadras y tiene datos sobre la información que tiene el jugador sobre el oponente.
- **Destroyer**: segundo tipo de unidad de infantería, más cara que el marine, pero con más vida. Es útil contra los vehículos, pero débil contra los snipers.
- **eSport**: deporte electrónico. Se dice de los videojuegos que se juegan a alto nivel competitivo, con torneos, premios y competiciones.
- **Expansión**: referente a las nuevas bases que se crean en un juego de estrategia sin contar la base principal.
- **Factory**: tipo de edificio empleado para crear las unidades blindadas.
- **GUI Manager**: Manager encargado de comunicar con la interfaz.
- **Indie**: juego independiente, normalmente desarrollado por un equipo pequeño sin Publisher.
- **Macro**: referente en un juego de estrategia a la gestión de los recursos y de la economía.

- **Main Building:** tipo de edificio donde se crean los trabajadores, en el llevan las materias recogidas.
- **Manager:** parte del código que se encarga de una serie de funciones con un propósito parecido.
- **Marine:** tipo de unidad básica, barata pero poco ataque contra infantería y vehículos.
- **Micro:** referente en un juego de estrategia al manejo individual de las unidades que el jugador posee.
- **Mods** (modificaciones): como su nombre indica son modificaciones que se hacen a los propios juegos, normalmente por parte de la comunidad de usuarios.
- **Recurso:** moneda de cambio en los juegos de estrategia, con él se crean unidades, edificios o mejoras.
- **Resource Manager:** Manager encargado de la gestión de los recursos, del manejo de los trabajadores y gestión de las tareas. En él se decide que se crea o construye en cada momento, es el Manager más complejo de todos.
- **Scout:** primera unidad blindada, utilizada sobre todo para explorar. Es rápida, pero no tiene mucho ataque ni vida.
- **Scoutear** (exploración): se dice del proceso de explorar el mapa en un juego de estrategia para obtener información sobre el oponente.
- **Sniper:** tercer tipo de unidad de infantería, la más cara de todas, pero muy efectiva contra otra infantería, no tanto contra vehículos.
- **State machine** (máquina de estados): método de programación por el cual se abstraen los distintos comportamientos de la unidad en sí.

Esta elige cuando cambia de estado, ejecutando una sección u otra del código según el estado elegido.

- **Supply:** tipo de edificio, básico que crea suministros para poder crear más unidades. Cuantos más suministros tengas más unidades se pueden crear.
- **Tank:** Unidad blindada. Es la más potente de todas, pero también la más cara.
- **Triple A:** se dice de los juegos con grandes presupuestos y *publishers* (editores), normalmente creados por las grandes empresas.
- **Turret:** torre defensiva. Edificio que ataca a las unidades enemigas que estén en su rango de acción.
- **Unit Manager:** Manager encargado de las unidades, de administrarlas y de darles acceso a los otros managers de estos datos.
- **Parsear:** proceso automático de leer un texto y obtener datos de dicho texto, que se guardan en distintas variables.
- **Micrear:** manejar, en un juego de estrategia a tiempo real, las unidades de forma individual.

10.2. Abreviaturas

2D	Dos dimensiones
3D	Tres dimensiones
A	Academy
AoE	Age of Empires
B	Barracs
C&C	Command and Conquer
Com	Combat
DotA	Defense of the Ancient
Eco	Economy
F	Factory
FPS	(Frames Per Second) Tasa de refresco por segundo
IA	Inteligencia Artificial
LoL	League of Legends
MB	Main Building
MMORTS	Masive Multiplayer Online Real-Time Strategy
MOBA	Multiplayer Online Battle Arena
PC	(Personal Computer) Computadora personal
RTS	(Real-Time Strategy) Juego de estrategia a tiempo real
RTT	Real Time Tactics
S	Supply
SC	Starcraft
T	Turret
TBS	Turn Base Strategy
TBT	Turn Base Tactics

11. BIBLIOGRAFÍA Y REFERENCIAS

Buvkland, M. (2005). *Programming Game AI by Example*. Estados Unidos de América: Wordware Publishing, Inc.

Gamma, E. & et al. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Estados Unidos de América: Addison Wesley

Holmström, F. (2014). *FogOfWar-Desktop: Fog of war for desktop devices*. GitHub. Recuperado de:

<https://github.com/fholm/unityassets/tree/master/FogOfWarDesktop>

Rabin, S. (2002). *AI Game Programming Wisdom (1ª Edición)*. Estados Unidos de América: Charles River Media.

Rabin, S. (2004). *AI Game Programming Wisdom 2 (1ª Edición)*. Estados Unidos de América: Charles River Media.

Rabin, S. (2014). *Game AI Pro*. Estados Unidos de América: CNC Press.