



Escuela
Politécnica
Superior

Gestión y manejo de comportamientos grupales emergentes de entidades en videojuegos



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:

Borja Pozo Wals

Tutor/es:

Francisco José Gallego Durán

Octubre 2020



Universitat d'Alacant
Universidad de Alicante

Gestión y manejo de comportamientos grupales de entidades en videojuegos

Autor

Borja Pozo Wals

Tutor/es

Francisco José Gallego Durán

Departamento de Ciencia de la Computación e Inteligencia Artificial



GRADO EN INGENIERÍA MULTIMEDIA



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, 23 de mayo de 2021

Resumen

El proyecto consiste en una herramienta para simular escenarios bélicos al estilo de los que podemos encontrar en juegos del género *Real Time Strategy (RTS)* como son ‘*They are Billions (TaB)*’ o la saga ‘*Age of Empires (AoE)*’, desarrollado completamente en C++ para Personal Computer (PC).

La “demo” se compone de un escenario, una serie de unidades controladas por el jugador que conforman el ejército bajo sus órdenes y por último, una serie de objetivos enemigos que deberemos abatir. Una vez alcanzado el reto propuesto, el juego termina dando la opción al jugador entre repetir el nivel o volver al menú principal.

Justificación y objetivos

A lo largo de la carrera son muchas las asignaturas que requieren y en las que se desarrollan habilidades relacionadas con la programación en diversos lenguajes y usos, pero no es hasta el tercer año que se me presenta la oportunidad de desarrollar un videojuego completo.

Durante la asignatura de ‘Fundamentos de los Videojuegos’ tuve por primera vez la experiencia de enfrentarme al desafío que supone crear un videojuego desde cero. Fue en ese momento cuando me di cuenta de que a lo que me quería dedicar es a desarrollar juegos de forma profesional. A lo largo del cuarto curso y junto a los demás integrantes de ‘Sunlight Studio’ desarrollamos ‘Cyborgeddon’. Este proyecto también contribuyó a que decidiera que es a los videojuegos a lo que quiero dedicar mi futuro.

A lo largo de mi vida he jugado a una considerable cantidad de juegos entre los cuales se puede apreciar una inclinación por los juegos de aventura y exploración, los Rol Play Game (RPG) y los RTS o de estrategia en general. Precisamente por esto que me hace especial ilusión desarrollar un juego de uno de estos géneros.

Teniendo en cuenta esto la elección de desarrollar un juego del tipo RTS para la realización del Trabajo Final de Grado (TFG) se debe a que, bajo mi criterio, es el género con mayor potencial para ayudarme a mejorar mis habilidades como programador, ya que se basa en unas mecánicas con las que no he trabajado con anterioridad.

Una vez dicho esto, los objetivos planteados para el proyecto son los siguientes:

- Aumentar mis conocimientos de C++
- Comprender mejor el funcionamiento del PC
- Aprender nuevas técnicas de Inteligencia Artificial (IA)
- Profundizar en la arquitectura Entity-Component-System (ECS)
- Desarrollar un producto mediante el cual mostrar mis habilidades

Agradecimientos

*A mi madre Gabriela,
por el increíble esfuerzo que realiza cada día por mí.*

*A mi hermana Raquel,
por haber sido y ser un faro para mí.*

*A mi buen compañero Jorge Espinosa,
por nuestra convivencia estos años.*

Para saber hablar es preciso saber escuchar

Plutarco.

Índice general

| | |
|--|-----------|
| 1 Introducción | 1 |
| 2 Marco teórico | 3 |
| 2.1 Age of Empires II: The Age of Kings | 3 |
| 2.2 They Are Billions | 6 |
| 2.3 Sea Salt | 9 |
| 2.4 Técnicas de inteligencia artificial | 11 |
| 3 Documento de Diseño del Juego (GDD) | 15 |
| 3.1 Descripción general | 15 |
| 3.2 Mecánicas | 15 |
| 3.3 Unidades | 17 |
| 3.4 Controles | 17 |
| 3.5 Pantallas | 18 |
| 3.6 Estados del juego | 19 |
| 3.7 Niveles | 20 |
| 3.8 Escenarios | 21 |
| 3.9 Sistema de cámaras | 21 |
| 3.10 Mínimo producto viable | 22 |
| 3.11 Actualización: 13/04/21 | 22 |
| 3.12 Actualización: 06/05/21 | 23 |
| 4 Metodología | 25 |
| 4.1 Metodología | 25 |
| 4.2 Estructura de una iteración | 26 |
| 4.3 Herramientas utilizadas | 27 |
| 5 Desarrollo y fases del producto | 29 |
| 5.1 Fase 0: Definición del proyecto y primeros pasos | 29 |
| 5.2 Fase 1: Producto mínimo viable | 36 |
| 5.3 Fase 2: Últimos pasos | 44 |
| 6 Conclusiones | 51 |
| 6.1 Lecciones aprendidas | 52 |
| 6.2 Trabajos futuros | 52 |
| Bibliografía | 56 |

Índice de figuras

| | | |
|------|--|----|
| 2.1 | Carátula del juego original. | 4 |
| 2.2 | Ciudad siendo asediada. | 5 |
| 2.3 | Descripción Estructura. | 5 |
| 2.4 | Ejemplo condición. | 5 |
| 2.5 | Ejemplo condición. | 6 |
| 2.6 | Ejemplo constantes. | 6 |
| 2.7 | Imagen promocional del juego. | 6 |
| 2.8 | Ejemplo de elementos tecnológicos del juego. | 7 |
| 2.9 | Infectado gigante | 8 |
| 2.10 | Ejemplo de captura global del nivel. | 8 |
| 2.11 | Menú principal | 10 |
| 2.12 | Criaturas de ‘Dagon’ | 10 |
| 2.13 | Separation behavior | 12 |
| 2.14 | Alignment behavior | 12 |
| 2.15 | Cohesion behavior | 12 |
| 3.1 | MockUp Seguimiento de la marca del jugador. | 15 |
| 3.2 | MockUp unidades desordenada. | 16 |
| 3.3 | MockUp unidades en anillo. | 16 |
| 3.4 | MockUp inicio. | 18 |
| 3.5 | MockUp carga. | 18 |
| 3.6 | MockUp juego. | 18 |
| 3.7 | MockUp pausa. | 18 |
| 3.8 | MockUp victoria. | 19 |
| 3.9 | MockUp derrota. | 19 |
| 3.10 | MockUp flujo de ejecución. | 19 |
| 3.11 | MockUp nivel 0. | 20 |
| 3.12 | MockUp nivel 1. | 20 |
| 3.13 | Ejemplo escenario con vegetación. | 21 |
| 4.1 | Logo de Trello | 26 |
| 4.2 | Logo de Toggl | 26 |
| 4.3 | Diagrama de iteración | 26 |
| 5.1 | Posibles resultados del bucle de juego. | 30 |
| 5.2 | Ejemplos entero con coma fija. | 32 |

| | | |
|------|---|----|
| 5.3 | Movimiento rectilíneo uniforme. | 32 |
| 5.4 | Movimiento rectilíneo acelerado. | 32 |
| 5.5 | Diagrama Arrive Behaviour. | 33 |
| 5.6 | Esquema fuerzas del flocking | 34 |
| 5.7 | Esquema algoritmo de Bresenham. | 34 |
| 5.8 | Esquema acceso a vector de punteros. | 37 |
| 5.9 | Esquema acceso a vector de objetos. | 37 |
| 5.10 | Marcador dirección entidad. | 39 |
| 5.11 | Formación anillo por rango de cohesión | 40 |
| 5.12 | Herramienta depuración en tiempo de ejecución | 41 |
| 5.13 | Mini Mapa. | 41 |
| 5.14 | Demostración sistema de disparos. | 42 |
| 5.15 | Formación anillo con trazado de rayos. | 42 |
| 5.16 | Comparación resultados división. | 45 |
| 5.17 | Arrive behaviour en la cámara | 46 |
| 5.18 | Cambios en el minimapa | 47 |
| 5.19 | Selector de comportamiento y formación | 47 |
| 5.20 | Mensajes estáticos nivel | 50 |

Índice de tablas

| | | |
|-----|--|----|
| 3.1 | Unidades y estadísticas | 17 |
| 5.1 | Leyenda finalización de tareas. | 29 |
| 5.2 | Resumen fase 0 | 30 |
| 5.3 | Resumen fase 1 | 36 |
| 5.4 | Fuerzas de separación en función de la distancia | 43 |
| 5.5 | Resumen fase 3 | 44 |
| 5.6 | Valores arquero vs soldado | 48 |
| 5.7 | Valores soldado vs arquero | 49 |

Índice de Acrónimos

A lo largo del documento serán utilizadas una serie de abreviaturas con el fin de hacer más cómoda su lectura. Todos los términos están indicados a continuación:

- **TFG:** Trabajo Final de Grado
- **PC:** Personal Computer
- **RTS:** Real Time Strategy
- **RPG:** Rol Play Game
- **IA:** Inteligencia Artificial
- **NPC:** Non-Player Character
- **AoE:** Age of Empires
- **TaB:** They are Billions
- **ECS:** Entity-Component-System
- **GDD:** Game Design Document
- **MVP:** Mínimo producto viable

1 Introducción

Este proyecto aborda el desarrollo de un simulador de batallas para Personal Computer (PC) desarrollado en *Linux*, escrito en C++ y haciendo uso de librerías escritas en C como TinyPTC (6) o DearImGui (10).

El prototipo consiste en una demo del género Real Time Strategy (RTS) compuesta por un escenario a través del cual tendremos que comandar y luchar junto a nuestro ejército con el fin de abatir al ejército rival. El escenario estará compuesto por estructuras y obstáculos que deberemos sortear para alcanzar nuestro objetivo.

A nivel técnico el proyecto supone el desafío de desarrollar una Inteligencia Artificial (IA) grupal para los Non-Player Character (NPC) basada en el uso de técnicas como los *Steering behaviours* y el *Flocking* con el fin de emular un comportamiento coordinado entre las distintas entidades. El objetivo es crear una versión inicial sencilla mediante la que profundizar en los conceptos en los cuales se basan las mencionadas técnicas con el fin de desarrollar una versión más compleja y específica para nuestro proyecto de forma que se adapte de la mejor forma posible a nuestras necesidades.

Por otro lado, un objetivo adicional fuera de la demo como tal es realizar un *port* a *Windows* con el fin de poder mostrar los resultados en ambos sistemas operativos.

2 Marco teórico

Cuando uno busca desarrollar un producto, conviene investigar qué se ha realizado previamente en entregas similares, ver que técnicas se han utilizado, con qué problemas se han encontrado y cómo los solventaron. También resulta interesante ver qué mecánicas han perdurado y qué innovaciones se han ido introduciendo con el paso del tiempo.

A continuación vamos a detallar aspectos de interés encontrados en diversas entregas que nos servirán de referentes.

2.1. Age of Empires II: The Age of Kings

En primer lugar encontramos el juego '*Age of Empires II: The Age of Kings*'. En él encontraremos una serie de campañas a través de las cuales encarnaremos a personajes históricos como 'Juana de Arco' o 'William Wallace' y los acompañaremos en sus conquistas y batallas más icónicas. Además de grandes batallas encontraremos el deber de gestionar nuestra facción, este trabajo nos requerirá recoger recursos a lo largo del mapa mientras desarrollamos nuestras ciudades y unidades.

El desarrollo tecnológico se divide en cuatro etapas históricas: Alta Edad Media, de los Castillos, Feudal e Imperial. Cada una de estas etapas trae consigo una serie de unidades, edificaciones e investigaciones nuevas que nos proporcionaran escenarios más complejos a nivel estratégico. Para pasar de una etapa a otra no es necesario completar al 100 % las opciones desbloqueadas, solo unos requisitos mínimos.

El juego cuenta con un total de 35 civilizaciones o facciones. Todas ellas tienen acceso al mismo árbol tecnológico pero cuentan con ligeras diferencias, pues cada una tiene una o dos tecnologías únicas y características especiales en una unidad militar¹.

Todas la unidades militares² de las que podemos disponer cuentan con una serie de características que las hacen más o menos fuertes en función del objetivo al que se enfrenten. A este respecto, tenemos el ejemplo de las máquinas de asedio las cuales son más fuertes contra estructuras pero más débiles contra infantería, o las unidades a caballo que son fuertes contra arqueros e infantería pero débiles frente a lanceros.

Este tipo de mécanicas dotan al juego de una importante componente táctica en el

¹Normalmente alguna cualidad mejorada en relación a las demás como puede ser la velocidad o daño.

²Lista de unidades: [https://ageofempires.fandom.com/wiki/Units_\(Age_of_Empires_II\)](https://ageofempires.fandom.com/wiki/Units_(Age_of_Empires_II)).

manejo de las unidades que debemos dominar si queremos completar los diferentes niveles de forma satisfactoria.

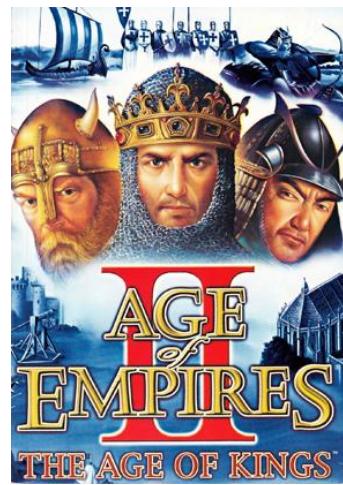


Figura 2.1: Carátula del juego original.

En lo referente a edificaciones³ podemos encontrar distintos tipos según su aporte al jugador. El edificio principal es el ‘Ayuntamiento’, que nos permitirá crear colonos para que trabajen recogiendo recursos y/o ayudando en la construcción de nuevas estructuras. En segundo lugar tenemos edificios como el ‘Cuartel’ o el ‘Campo de Tiro’, que nos permiten crear unidades militares, así como edificios como la ‘Herrería’ la cual nos permitirá realizar investigaciones y crear unidades de asedio.

Por último podemos encontrar estructuras de temática religiosa como el ‘Monasterio’, otras destinadas a aumentar la productividad en la explotación de recursos naturales como el ‘Molino’ y defensivas como la ‘Muralla’.

En el género *RTS* hay una tendencia al uso de una perspectiva isométrica con cámara fija, que solo podemos modificar haciendo *zoom* o desplazándola por el entorno. Esto seguramente se deba a que es la configuración que mejor nos permite observar el mapa y todas las unidades que hay en él. Además, el hecho de que sea fija nos libra de la problemática de ir ajustando la cámara según la parte del mapeado en la que estemos. Cabe esperar el mapeado y los elementos visuales del juego también hayan sido diseñados teniendo esto en cuenta.

Como podemos ver en el artículo de Pritchard sobre el desarrollo de *Age of Empires (AoE) 2*, la IA del juego está desarrollada completamente con un lenguaje de *scripting* desarrollado por ellos. De esta forma el código consiste en una serie de constantes y variables definidas desde un inicio y mediante condicionales realizar las acciones y cambios pertinentes. Gracias a un usuario de ‘GitHub’ podemos comprobar en las imágenes

³Lista de edificios: [https://ageofempires.fandom.com/wiki/Buildings_\(Age_of_Empires_II\)](https://ageofempires.fandom.com/wiki/Buildings_(Age_of_Empires_II)).



Figura 2.2: Ciudad siendo asediada.

el código usado en el juego y conocer en detalle cómo está elaborado.

```
(defrule
  (CONDITION 1)
  (CONDITION 2)
=>
  (ACTION 1)
  (ACTION 2)
)
```

Figura 2.3: Descripción Estructura.

```
(defrule
  (can-train villager)
=>
  (train villager)
)
```

Figura 2.4: Ejemplo condición.

Fuente: guía de scripting para IA de AoE 2 de Redmechanic.

Además de estos *scripts* el juego utiliza un sistema de *pathfinding* en dos niveles. El primero de los algoritmos calcula la ruta a seguir por las unidades sin tener en cuenta los elementos triviales como las demás unidades, mientras que la segunda pasada calcula el camino en tramos específicos con el fin de esquivar otras unidades, edificaciones, etc...

En lo referente al movimiento de las unidades, tanto por la forma en la que está programada la IA como por la sensación que dan al jugar, es posible que se trate de un “*Goto*” clásico y sencillo para evitar realizar muchos cálculos.

Además, con el tiempo el juego se ha convertido en un objetivo para “*Speedrunners*” y con una escena competitiva que se basa en ser el más rápido jugando⁴ por lo que una reacción instantánea de las unidades es un factor importante para estos jugadores.

⁴En los enfrentamientos de alto rango se llegan a medir las acciones por minuto (APM).

```
(defrule
  (taunt-detected any-ally 33)
  (game-time > 300)
=>
  (acknowledge-taunt this-any-ally 33)
  (chat-to-allies-using-id 22157)
  (set-goal train-civ-goal 1)
)
;
```

```
;Goals
(defconst increase-town-size-goal 1); this increases town size
(defconst attack-goal 2)
(defconst strategy-goal 3)
(defconst unit-goal 4)
(defconst train-civ-goal 5);1=train villagers, !=1 no villager
(defconst control-goal 6); 6 = allow to be shot, 7 = shot, als
(defconst anti-cavalry-threat-goal 7)
```

Figura 2.5: Ejemplo condición.

Figura 2.6: Ejemplo constantes.

Fuente: repositorio de '*GitHub*' del usuario Andygmb.

2.2. They Are Billions

En segundo lugar podemos encontrar el juego '*They are Billions (TaB)*', que se localiza en un mundo postapocalíptico infestado de zombis donde quedan muy pocos supervivientes. El juego cuenta con dos modos:

En primer lugar encontramos el modo **horda**, en el que deberemos sobrevivir a sucesivas oleadas de zombis que serán cada vez más grandes y con unidades más poderosas. Además, deberemos mejorar nuestra colonia y sus defensas para poder lograr pasar las rondas.

En segundo lugar tenemos el modo **campaña**, que consiste en una serie de misiones con el fin de ampliar el tamaño de nuestro imperio a la vez que conocemos su historia y personajes clave.



Figura 2.7: Imágen promocional del juego.

En '*TaB*' podemos ver cómo se reutilizan una serie de características propias del género como pueden ser el uso de una perspectiva isométrica con cámara fija o el uso de un “árbol tecnológico” por el cual deberemos escalar si queremos desbloquear nuevas unidades y/o mejorar las ya disponibles.

Si nos fijamos en las construcciones⁵ disponibles en el juego, encontramos como se sigue la línea de un “Ayuntamiento” como núcleo de nuestra base, el cual nos permitirá conseguir trabajadores y nos dará acceso a las primeras mejoras. Por otro lado, encontramos las edificaciones destinadas a conseguir recursos, reclutar nuevas tropas y/o defender nuestra colonia.

En este juego se introduce como novedad el recurso de la ‘energía’, el cual será utilizado para mantener nuestras maquinarias en funcionamiento además de determinar cuánto y dónde podremos construir. Para ganar energía y ampliar el terreno disponible el jugador deberá construir ‘torres de Tesla’ (o sus respectivas mejoras).

A diferencia de en ‘AoE’, en ‘TaB’ encontramos la aparición de solamente dos facciones: ‘El Nuevo Imperio’, la facción del jugador, la cual representa una serie de colonias que tendremos que desarrollar a lo largo de los niveles, y en el otro lado, encontramos las infinitas hordas de zombis que tratarán de exterminar a la raza humana.

Cada una de las facciones tiene sus propias unidades. El jugador contará con 7 unidades distintas entre las que podemos encontrar desde unas rápidas y sigilosas exploradoras hasta unidades equipadas con lanzallamas o montados en robots de combate.⁶

Las hordas enemigas estarán compuestas en su mayoría por unidades débiles que de forma eventual serán acompañadas por gigantes y/o otras unidades especiales.⁷



Figura 2.8: Ejemplo de elementos tecnológicos del juego.

A lo largo del juego podemos encontrar funciones interesantes e innovadoras como la pausa táctica, la cual nos permitirá visualizar detenidamente el estado del mapa sin tener que preocuparnos por no estar atendiendo algunos posibles eventos, como ataques a nuestras tropas por parte del enemigo.

En juegos anteriores como los ‘AoE’ es fácil encontrarnos en la situación de tener trabajadores en la ciudad que están pausados sin tareas durante un tiempo indeterminado de

⁵Lista de edificios: <https://they-are-billions.fandom.com/wiki/Category:Buildings#Buildings>.

⁶Lista unidades imperiales: <https://they-are-billions.fandom.com/wiki/Category:Units>.

⁷Lista unidades zombi: <https://they-are-billions.fandom.com/wiki/Category:Infected>.



Figura 2.9: Infectado gigante

juego, no poder encontrar cuáles son e ir pensando en otra ocupación para ellos por estar atrapados en refriegas con otros jugadores. Con este tipo de mecánicas estas situaciones se solventan en mayor o menor medida y permiten al jugador tomarse el tiempo que necesite para pensar las acciones que quiere realizar.

Otra adición interesante es la posibilidad de hacer una captura de todo el mapeado explorado en el nivel pudiendo así mostrar el avance de nuestra ciudad y alrededores, además de la inmensidad de las oleadas de zombis que nos irán atacando.



Figura 2.10: Ejemplo de captura global del nivel.

En lo refente a la IA, podemos observar cómo intentan replicar la forma de actuar del zombi común. Como podemos leer en la entrevista de Sucasas a ‘Jesús Arribas’ y ‘Miguel Corral’⁸ para el portal ‘Xakata’, cada zombi cuenta con mecanismos para detectar a los humanos, ya sea por divisarlos visualmente o escucharlos moverse. Además si un compañero cercano ha encontrado un objetivo, este le seguirá siguiendo una mentalidad de manada a la que se irán sumando los zombis en su recorrido.

Mientras los infectados estén sin un objetivo específico, pueden darse dos escenarios:

⁸Responsables principales de Numantian Games y desarrolladores del juego.

el primero nos muestra enemigos dispersos por el mapa esperando a que el jugador se aventure a explorar esa zona y sorprenderle con un ataque. El otro nos trae las oleadas que aparecen en los límites del mapa las cuales tienen como objetivo el centro de nuestra base. Si los enemigos estáticos por el mapa divisan la horda se unirán a ella aumentando así su poder, lo que sin duda alienta al jugador a “limpiar” el mapa con el fin de aliviar estas acometidas.

Como la intención de la IA es la de replicar la forma de “no pensar” de los zombis, en el momento en que unidades humanas se presenten delante de las hordas de enemigos estos se dispondrán a perseguir a estas personas, perdiendo así el foco en nuestra base. Esto abre las puertas a poder jugar con el “*aggro*” de los zombis de forma estratégica.

En paralelo a las declaraciones de los *devs*, parte de la comunidad discutía sobre si la IA del juego hacía trampas con tal de incrementar la dificultad, a lo que el usuario Amordron comenta las declaraciones de los desarrolladores y niega que la máquina haga trampas. Para respaldar sus declaraciones nos redirige a un *post* de ‘Reddit’ donde Pikachunet nos muestra capturas de un posible editor de niveles del juego en el que se nos muestran multitud de opciones, entre las que encontramos la cantidad de enemigos, cómo se disponen en el mapa, o si tendrán esa iniciativa de ir al centro de la base.

2.3. Sea Salt

Como podemos observar en los anteriores referentes, dentro del género RTS da la sensación de que todos los recursos destinados a IA estaban reservados para la toma de decisiones y en gestionar la civilización, dejando el desplazamiento de las unidades y el combate como algo trivial.

Cabe destacar que el género nació cuando no había casi capacidad de procesamiento y, con el paso del tiempo, el hecho de intentar conservar su esencia ha contribuido a unas futuras experiencias de juego un “poco básicas”.

Por otro lado, hay juegos que han aportado experiencias y mecánicas nuevas, ya sea combinando géneros y/o incluyendo técnicas más modernas que nos ofrecen resultados distintos. Este es el caso de ‘Sea Salt’, juego que combina estrategia, acción y cartas en el que encarnaremos a ‘Dagon’ una antigua deidad marina la cual planea castigar a la humanidad por haberse opuesto a sus designios.

El juego se compone de una serie de niveles que representan distintas partes de la ciudad donde se desarrolla la historia. Cada nivel tiene a su vez distintas salas donde encontraremos diversos obstáculos y enemigos que abatir. Para abrirnos paso por el nivel manejaremos al ejército de ‘Dagon’, compuesto por diversas criaturas con sus características únicas. A lo largo de los niveles encontraremos altares donde podremos invocar nuevas unidades para aumentar el grueso de nuestras filas.

En este caso los puntos que más nos interesa tratar es el uso de ‘*Flocking*’ y el control



Figura 2.11: Menú principal

de las unidades que controla el jugador.

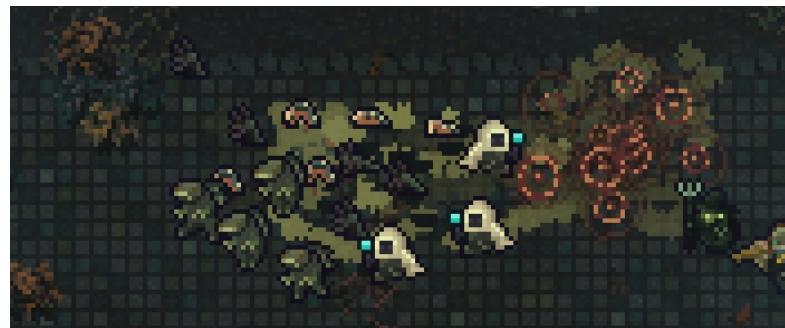


Figura 2.12: Criaturas de 'Dagon'

En cuanto a la jugabilidad y los controles, el jugador no manejará de forma explícita las criaturas del ejército, sino que, mediante las teclas ‘w’, ‘a’, ‘s’, ‘d’ moverá una marca por el mapa la cual será seguida por las unidades. Además, con la ‘barra espaciadora’ las unidades se lanzarán al ataque de los enemigos en su rango de acción, aparentemente al más cercano. Por último encontramos la tecla ‘ctrl’ que nos permitirá hacer que las unidades se reagrupen en la marca del jugador y el ‘shift’ que hará que todas las unidades avancen a la misma velocidad.

Mediante estas acciones tendremos que arreglárnoslas para matar a las unidades enemigas y esquivar sus ataques, sobre todo en las peleas con jefes al final de los niveles.

Por otro lado encontramos el uso de *'Flocking'* el cual aporta un toque “errático” en el desplazamiento de las unidades que, junto a los controles, juega mucho con la temática de monstruos/criaturas malvadas.

La técnica ayuda a darle vida y sensación de individualidad a cada una de las unidades, a la vez que refuerza la visión de conjunto al ver como las unidades se encuentran en

todo momento actuando de la misma forma.

2.4. Técnicas de inteligencia artificial

Como se menciona en la introducción, uno de los objetivos más importantes del proyecto recae en el desarrollo de una IA haciendo uso de las técnicas de '*Flocking*' y '*Steering behaviors*'.

Para ello nos basaremos principalmente en las explicaciones y ejemplos que podemos encontrar en el libro (Millington, 2009, ch. 3) donde de forma extensa y detallada se nos introduce en la teoría relacionada con los algoritmos y la forma en la que estos se estructuran e interactúan entre ellos. Para dar un poco de contexto sobre el tema resumiremos brevemente las ideas que se nos presentan a lo largo del capítulo dedicado a estas técnicas.

Los '*Steering behaviors*' pueden ser entendidos como una serie de algoritmos destinados a guiar la forma en la cual los NPC se desplazan por el escenario y/o interactúan con los distintos elementos que puedan encontrarse en la escena. Su función es la de crear movimientos complejos a base de una combinación de movimientos y/o acciones simples, un ejemplo común puede ser la acción de perseguir a un objetivo mientras se sortejan obstáculos en el proceso.

En este caso no tendríamos una función llamada “*persigue-enemigo-mientras-esquivas()*” que se encargue de todo el trabajo, sino que tendremos el cálculo de la velocidad y dirección necesarias para alcanzar el objetivo, la compropación para saber si hay algún tipo de obstáculo por el camino y la rectificación de la trayectoria en caso de haberlos cada uno por su lado. Es el resultado de todos estos pasos lo que define el movimiento final.

Esta forma de estructurar y formar actividades complejas en base a acciones más simples nos permite reutilizar y jugar con los diferentes comportamientos permitiéndonos crear con ellos un amplio espectro de resultados.

En lo referente al '*Flocking*' podemos observar como en esencia es lo mismo que los '*Steering behaviors*' pero añadiendo factores y/o componentes grupales, el origen del modelo lo podemos encontrar en las publicaciones de Reynolds (1986) donde se nos introduce el concepto de "*Boid*" como entidad genérica que simula su comportamiento bajo este algoritmo. Además, se nos introducen los tres comportamientos básicos en los que se basa la técnica para generar el movimiento emergente, que son:

La **separación** que cada *Boid* mantendrá entre las demás entidades en su vencidad, gracias a la cual evitaremos solapamientos y respetaremos el espacio y movimiento de las demás entidades.

Por otro lado podemos encontrar el **alineamiento** de la dirección del movimiento propio con la dirección de las entidades más cercanas, de esta forma conseguimos un

movimiento armónico entre los *boids* y produciremos una sensación de coordinación entre ellos.

Por último encontramos la **cohesión**, encargada de mantener a las entidades cercanas juntas para crear esa sensación de grupo que buscamos con el algoritmo.

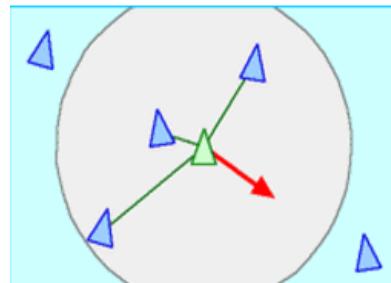


Figura 2.13: Separation behavior

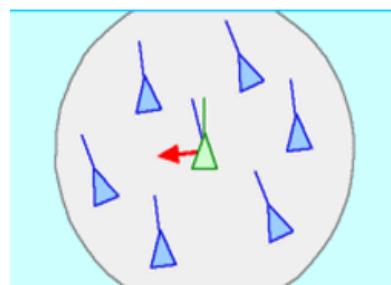


Figura 2.14: Alignment behavior

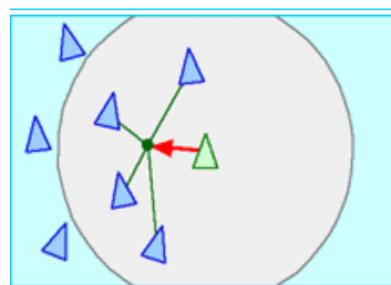


Figura 2.15: Cohesion behavior

Además de esto, podemos ver en el artículo de ‘Raynolds’ cómo a lo largo de los años se ha ido modificando y ampliando el algoritmo con el fin de añadir variaciones en el comportamiento y/o introducir más factores influyentes en la decisión de los *Boids* como puede ser el olor de determinada entidad/es y/o escenario.

Esto sin duda es gracias a la versatilidad que nos proporciona el uso de los ‘Steering

behaviors' y jugar con la importancia de las distintas componentes a la hora de hacer la toma de decisiones.

3 Documento de Diseño del Juego (GDD)

3.1. Descripción general

El juego que se pretende desarrollar se basa en el apartado de navegación por el mapa y combate típicos en los RTS. A lo largo de los distintos niveles, el jugador deberá hacer frente a distintos desafíos como pueden ser: escoltar con sus unidades de un punto del mapa a otro a un personaje importante y/o eliminar a todas las entidades enemigas del escenario.

3.2. Mecánicas

Durante el juego podremos ejecutar una serie de órdenes sobre nuestras unidades para cambiar su distribución, ordenarles que se desplazan y/o que ataquen a enemigos. Para el desplazamiento por el mapa, el jugador contará con un marcador en el mapa que podrá mover para que sus unidades lo sigan mientras les sea posible.

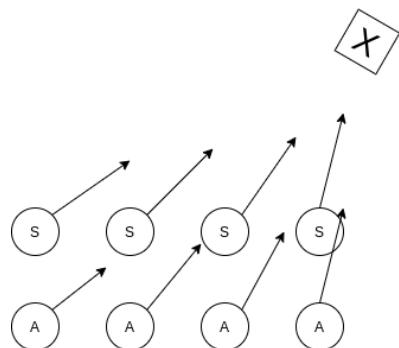


Figura 3.1: MockUp Seguimiento de la marca del jugador.

En cuanto a las posibles formaciones para nuestras unidades, encontramos las siguientes:

- **Sin formación:** las unidades no tendrán ningún parámetro de ordenación.

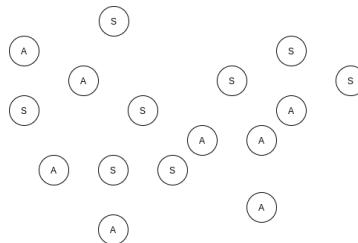


Figura 3.2: MockUp unidades desordenada.

- **En anillo:** formación circular alrededor de una unidad especial. En caso de no haber unidad escoltada el centro estará vacío.

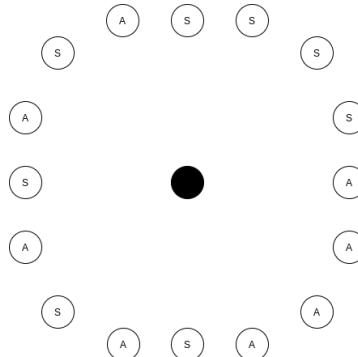


Figura 3.3: MockUp unidades en anillo.

Por último podemos ordenar a las unidades que mantengan una estrategia definida:

- **Huida:** en este modo las unidades seguirán el marcador sin pararse a pelear con unidades cercanas enemigas.
- **Agresivo:** en el momento en el que la tropa divisa un enemigo saldrá directo a atacarle, aunque se aleje esta le seguirá mientras siga en su rango de detección.

Aunque el jugador marque una estrategia y forma de plantear el combate, las unidades podrán decidir si seguir al jugador o revelarse. Si los soldados cercanos están siendo masacrados existe la posibilidad de que tengan miedo y huyan. Si las unidades enemigas nos superan en número la unidad puede decidir si permanecer en defensa en lugar de atacar de forma activa.

3.3. Unidades

Entre las unidades podemos encontrar dos arquetipos con características propias que nos permitirán crear variedad en las soluciones a la hora de superar el nivel.

Los tipos son los siguientes:

- **Soldado:** es la unidad más básica que podemos encontrar en el campo de guerra, está armado con una espada y posee estadísticas bajas.
- **Arquero:** va equipado con arco y flechas para atacar a distancia a sus rivales. Tiene menos resistencia que los soldados por lo que tendremos que protegerlos para asegurar su supervivencia.

| | Daño | Vida | Rango | Tiempo de ataque |
|---------|------|------|-------|------------------|
| Soldado | 7 | 24 | Melee | 2 seg. |
| Arquero | 4 | 14 | Largo | 3 seg. |

Tabla 3.1: Unidades y estadísticas

3.4. Controles

A la hora de jugar tendremos una serie de teclas asignadas a las acciones que el jugador puede realizar cuando interactúe con el juego. Para enumerarlas dividiremos las acciones en dos grupos, dependiendo de si son para navegar por los menús o si representan acciones durante el *gameplay*.

Para avanzar por los menús usaremos la tecla **Enter** una vez estemos sobre la opción deseada o para avanzar los mensajes explicativos del tutorial/informativos.

Las asignadas para jugar son las siguientes:

- **Flechas o W/A/S/D:** mediante estas teclas podremos desplazar el puntero por el mapa.
- **Barra Espaciadora:** con esta otra podremos ordenar atacar a nuestras unidades, si no encuentran una unidades enemiga en su rango se mantendrá en seguimiento del puntero.
- **Ctrl:** activa el modo huída y las unidades dejarán de combatir y seguirán el puntero.

- **Z:** activa la formación en anillo.
- **X:** desactiva toda formación.
- **Ratón:** interactuar con el *HUD*.

Tanto la **formación** como el **comportamiento** de las unidades se podrá cambiar desde un pequeño *HUD*, que se encuentra en la esquina inferior izquierda.

3.5. Pantallas

Al ejecutar el programa la primera pantalla que aparecerá será la de inicio. Se compone del título del juego, la fecha de lanzamiento, el nombre del desarrollador y un mensaje que nos indica que tenemos que pulsar la tecla *enter* para continuar, esta pantalla se mantendrá hasta que el jugador presione dicha tecla.

Una vez pulsado el botón indicado se procederá a cargar el juego mientras se muestra una barra que indica el progreso de cargado.

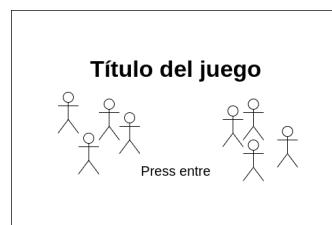


Figura 3.4: MockUp inicio.

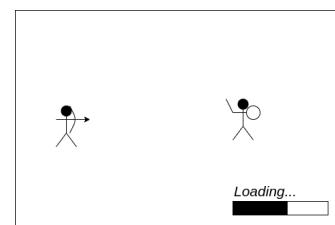


Figura 3.5: MockUp carga.

A continuación de la carga nos encontraremos con el escenario, donde se desarrollará el *gameplay* y nos mantendremos en esta pantalla hasta que termine el juego, ya sea por victoria o derrota del jugador.

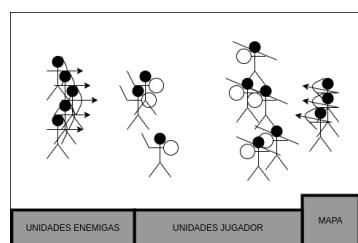


Figura 3.6: MockUp juego.

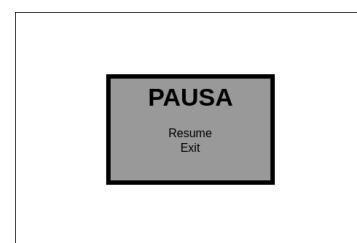


Figura 3.7: MockUp pausa.

El final de la partida nos trae dos posibles escenarios, la victoria y la derrota. Para cada uno saldrá su respectivo mensaje y pasado un momento se nos mandará automáticamente a la pantalla de inicio.

Una última pantalla con la que el jugador podrá interactuar es la del menú de pausa, en el cual se le dará la opción de salir o de volver a la partida, mientras esta pantalla este activa la acción en el juego se paralizará hasta que el jugador decida.

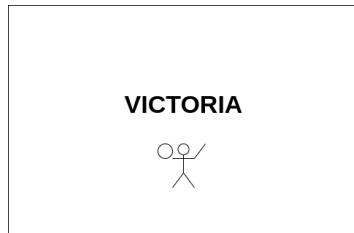


Figura 3.8: MockUp victoria.

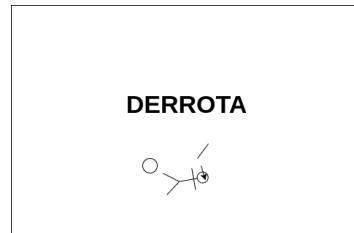


Figura 3.9: MockUp derrota.

3.6. Estados del juego

Una vez mostradas todas las posibles pantallas con las que podrá interactuar el jugador, es interesante dibujar un diagrama de flujo que plasme sus conexiones y posibilidades con el fin de crear una representación gráfica que sirva como esquema global.

Dicho esquema podemos encontrarlo en la figura siguiente:

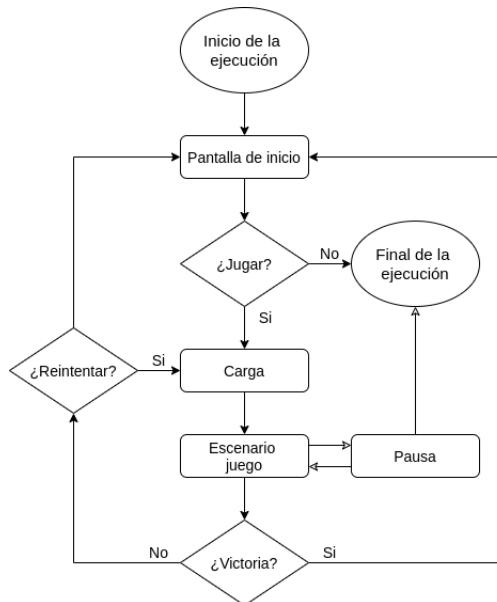


Figura 3.10: MockUp flujo de ejecución.

3.7. Niveles

El juego contiene actualmente 2 niveles, el primero se trata de un nivel introductorio en el que encontraremos grupos pequeños de unidades fáciles de abatir para que el jugador conozca cómo es el combate en el juego. Además, a lo largo del nivel pondrémos marcadores para que indiquen al jugador cómo jugar o *tips* para superar los niveles de forma satisfactoria. El nivel se supera matando a los enemigos.

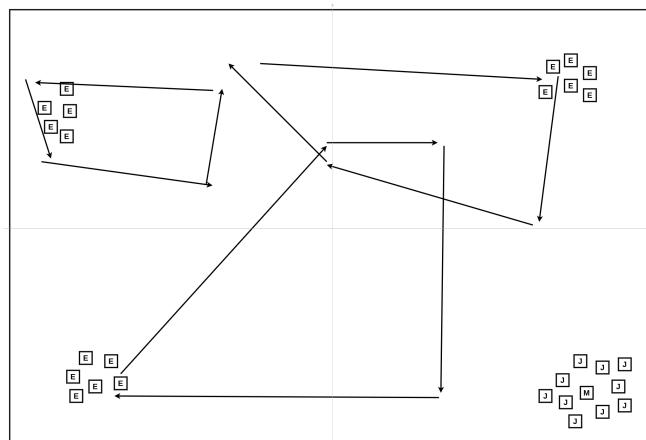


Figura 3.11: MockUp nivel 0.

En el segundo nivel se muestra otro de los objetivos posibles en el juego: alcanzar una zona de huída/meta más allá de las tropas enemigas. En este nivel encontraremos grupos más grandes de enemigos que nos pueden vencer y hacernos repetir el nivel, por ello podremos optar por intentar derrotarlos o buscar una ruta que nos evite el conflicto lo máximo posible.

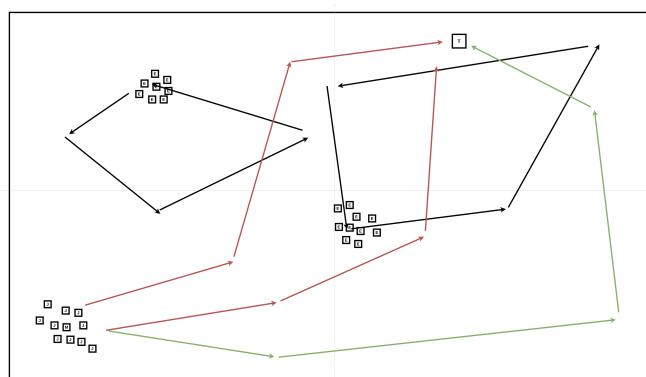


Figura 3.12: MockUp nivel 1.

3.8. Escenarios

A lo largo de los niveles la intención es que nos encontremos con distintos biomas como pueden ser praderas, bosque o desiertos similares a los que podemos encontrar en AoE II.



Figura 3.13: Ejemplo escenario con vegetación.

Además de imitar el estilo artístico el juego se desarrollará empleando una cámara aerea fija manteniendo una perspectiva isométrica que nos permitirá visualizar desde un punto elevado una gran porción del escenario. Esto lo haremos con el fin de dar al jugador la posibilidad de conocer lo máximo posible el territorio cercano y a la vez le libramos de preocuparse por situar la cámara para cada momento.

Al mantener una posición fija podemos situar todos los elementos en la escena de forma que no obstruyan la visión o molesten al jugador.

En las primeras versiones del proyecto el juego todos los elementos serán 2D y la vista será un plano picado, el final deseable será poder pasar a un entorno 3D.

3.9. Sistema de cámaras

Si quisiéramos mostrar todo el nivel por pantalla, tendríamos el problema de que los elementos quedarían dibujados con un tamaño demasiado pequeño si el nivel supera las dimensiones de la pantalla. Por ello, si queremos poder diseñar niveles con las dimensiones que queramos deberemos mostrar únicamente la fracción del nivel que ocupamos en cada momento.

En este caso, tendríamos una cámara que seguirá al puntero del jugador por el nivel y tendremos el minimapa para poder ubicarnos en el nivel.

3.10. Mínimo producto viable

Una vez planteadas todas las características deseadas para el videojuego, es acosenjable decidir qué partes del producto son vitales para mostrar la experiencia de juego deseable. El hecho de marcar estos objetivos nos ayudará a poder desarrollar una versión reducida del proyecto que nos permita dar la sensación de tener un producto acabado. Es importante realizar este proceso ya que por motivos internos o externos al equipo de desarrollo, siempre existen contratiempos que nos impidan conseguir todos los objetivos propuestos en el documento.

En cuanto a las etapas en la ejecución del juego, encontramos esencial el poder finalizar el nivel, ya sea por victoria o derrota. Acto seguido volver al comienzo del nivel, dejando a un lado un posible menú inicial y de pausa.

Una vez en partida el juego constará de un nivel único compuesto por: un escenario estático mínimo, un conjunto de unidades propiedad del jugador y otro perteneciente a la máquina.

Al inicio del juego el jugador se encontrará quieto en uno de los extremos del nivel y el enemigo estará patrullando por una zona designada, dejando así a elección del jugador la distribución y el momento exacto en el que comenzará la disputa.

En lo referente a las mecánicas y jugabilidad, es importante que el jugador sea capaz de poder mover a sus unidades por el escenario y ordenarles atacar. También sería deseable poder ajustar algunos parámetros del comportamiento de su ejercito para poder pasar el nivel de más de una forma.

Por otro lado, la IA debe ser capaz de detectar el acercamiento del jugador y responder a la ofensiva, dejando la capacidad de tomar la iniciativa y variar su estrategia para versiones más completas. Ambos ejércitos se compondrán de soldados y arqueros.

En lo referente al apartado visual, sería deseable tener sprites para unidades y escenario pero de ser necesario podemos mantener el sistema de *render* inicial.

3.11. Actualización: 13/04/21

- Corregido el movimiento lento de las entidades en algunos momentos cuando estaban en el radio de frenada debido a un cálculo erróneo en las divisiones con el tipo de dato en coma fija.
- Creación de una componente para las colisiones e integración en los sistemas pertinentes.
- Solucionado el problema de las balas que no colisionaban bien.
- Adicción de los *Colliders* al modo de depuración visual.

- Cambios ligeros en el sistema de eventos de las balas y creación y destrucción de entidades.
- Creación de un *HUD* para seleccionar comportamientos y formaciones de las unidades aliadas.
- Rediseño completo en la toma de decisión de las unidades aliadas y enemigas.
- Creación de un componente de BlackBoard para el seguimiento al pj v0.1.
- Sistema de cámaras y coordenadas de mundo: ahora las dimensiones del nivel son independientes del tamaño de la ventana, se usan los límites del nivel para no dejar pasar a las unidades, se hace *clipping* sobre las unidades que no están en la cámara, el minimapa ahora muestra la porción de nivel que se muestra.
- Se ha mejorado el sistema de entidades objetivo y los mensajes de muerte y ataque para cambiar de objetivo cuando el actual muera, además al morir una unidad se eliminan los demás mensajes asociados a ella.
- El componente de IA ya no contiene la ruta directamente, ahora contiene un iterador a su ruta y se ha creado el tipo de dato ruta con operadores sobrecargados para navegar por ella. También tenemos “caminos” diseñados pero sin incluir, los cuales son caminos no cíclicos.
- Se ha ampliado un poco el GDD con apartados sobre tareas pendientes y objetivos.
- En el diario de desarrollo se han sustituido códigos de ejemplo por mockups.

3.12. Actualización: 06/05/21

- Redacción problema en las divisiones de los enteros con coma fija.
- Redacción problema con las colisiones y solución.
- Redacción *HUD* v0.4.
- Redacción sistema de cámaras en el GDD y diario de desarrollo.
- Redacción cambios en las entidades: rutas y comportamientos.

4 Metodología

4.1. Metodología

La metodología llevada a cabo durante el proyecto es una creación propia basada en características copiadas de metodologías ágiles como el '**Scrum**' y adaptadas a la forma de trabajar propia. La idea es estructurar todo el proyecto en diversas iteraciones y durante estas marcarse objetivos y/o tareas con el fin de añadir funcionalidades completas.

El uso de **iteraciones** para dividir la carga de trabajo y agrupar tareas nos ayudará a conseguir acotar la duración de algunas tareas y marcarnos objetivos a corto/medio plazo. Además de separar por funciones el proyecto, las tareas que conllevan su desarrollo también deberemos analizarlas e intentar reducir su tamaño y carga lo máximo posible, lo que permite obtener una sensación de éxito de forma rápida y contribuye mantener una moral y motivación altas.

Para ilustrar cómo se elabora esta separación por subtareas, el objetivo que vamos a usar es “dibujar elementos por pantalla”. Como estamos en un proyecto basado en Entity-Component-System (ECS) rápidamente vemos que vamos a requerir como mínimo un componente de dibujado (*RenderCmp*) y un “*RenderSystem*”, que se encargará de recoger todos los elementos visuales y hacer las llamadas necesarias para dibujarlos. En caso de no tener una librería de dibujado, otra tarea podría ser el buscar una que sea capaz de realizar el trabajo requerido y/o implementar una propia ¹.

Para ayudar a la planificación del trabajo usaremos la herramienta ‘Trello’ ² la cual nos permitirá crear diversas listas según el ámbito o el estado de desarrollo de las distintas tareas que contendrán, cada tarea se ve representada con una tarjeta que puede tener una serie de subtareas asociadas dentro. Esto nos permitirá tener un registro de todas las labores que quedan por hacer en cada iteración y cuáles ya están terminadas completamente. Además, podremos conservar las listas de las tareas realizadas en iteraciones anteriores para futuras revisiones.

Otra herramienta de control que usaremos durante el desarrollo será ‘Toggl’ ³ la cual nos permitirá llevar a cabo un registro de las horas que dedicamos en cada tarea y agrupar tareas en función del campo de estudio o parte del desarrollador al que pertenecen.

¹Esto supondría una serie de tareas en base a la cantidad de funcionalidades que se requieran.

²Página de ‘Trello’: <https://trello.com/es>

³Página de ‘Toggl’: <https://toggl.com>



Figura 4.1: Logo de Trello



Figura 4.2: Logo de Toggl

4.2. Estructura de una iteración

Con el fin de trabajar de la mejor forma posible, a lo largo de cada iteración encontramos una serie de fases/tareas organizativas que guian el flujo de trabajo.

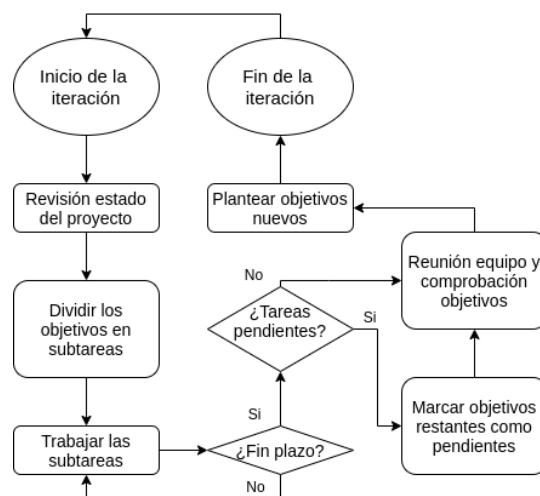


Figura 4.3: Diagrama de iteración

El inicio de la iteración trae consigo un análisis y revisado del proyecto, con la idea de que lo último implementado se encuentre en un estado aceptable y todo preparado para recibir las futuras adiciones al proyecto si es que las va a haber.

Una vez esté todo correcto procedemos al estudio y división de las futuras tareas, a la vez que las categorizamos según la temática o aspecto del proyecto que se trabaje. Con una visión global de todo el trabajo podemos organizar cómo vamos a abordar el desarrollo, analizar si tenemos dependencias entre tareas y acto seguido empezar. Por lo general suelen surgir inconvenientes o nuevas necesidades durante las iteraciones que nos harán rehacer, posponer o simplemente añadir algunas tareas, por lo que es posible tener que ampliar de forma puntual este apartado a lo largo de la iteración.

Cuando el apartado organizativo esté terminado (sin tener en cuenta contratiempos) lo único que queda por hacer es entrar de lleno en el desarrollo, esta fase se extenderá hasta que se termine el plazo de la iteración o se acaben los objetivos propuestos.

Conforme se acerca el final de la iteración, lo último es una reunión de control donde se comprueba el porcentaje del trabajo propuesto que ha sido completado, posibles problemas encontrados durante la iteración y que soluciones se han aportado (en el caso de haber encontrado una). Una vez terminada la discusión, se procede a marcar las nuevos objetivos , terminando así la actual iteración y comenzando la siguiente en caso de no finalizar todavía el desarrollo del proyecto.

4.3. Herramientas utilizadas

El proyecto desarrollado utiliza estas herramientas y está destinado a los siguientes sistemas:

- **Lenguaje:** C++17 y Makefile
- **Compilador Linux:** GCC v10.2.0
- **Compilador Windows:** MinGW64 v10.2.0
- **SO:** Windows 10 y Manjaro v20.2
- **Librerías:** Dear ImGUI v1.81

Tanto el producto desarrollado como la memoria están disponibles en las siguientes fuentes: en este repositorio de Github y en una entrada de Archive.org.

5 Desarrollo y fases del producto

A lo largo de esta sección comentaremos todo el proceso de desarrollo del prototipo y de la memoria. Las diferentes iteraciones del producto serán agrupadas en diversas fases con el fin de facilitar la lectura y el entendimiento de los objetivos planteados en cada momento. Además, se contarán de forma más extensa los puntos relevantes y objetivos en este proyecto.

| Terminado es esta iteración (vacío) | Terminado en otra iteración It.X | Sin definir cuando se terminará ∞ |
|--|-------------------------------------|---|
|--|-------------------------------------|---|

Tabla 5.1: Leyenda finalización de tareas.

5.1. Fase 0: Definición del proyecto y primeros pasos

| Iteraciones | % Completado | Terminado en |
|--|--------------|--------------|
| 0.1. Definir la idea. | 60 % | It.2 |
| 0.2. Iniciar desarrollo del prototipo Integrar TinyPTC y dibujar por pantalla. | 100 % | |
| 0.3. Adquirir hábito de escribir, preparar las herramientas y entorno de LaTex. | 50 % | It.2 |
| 0.4. Prototipado del movimiento usando <i>Steering behavior</i> . | 30 % | It.1 |
| 1.1. Creación tipo con coma fija. | 100 % | |
| 1.2. Busqueda de referentes. | 40 % | It.3 |
| 1.3. Corregir funcionamiento de los <i>Steering behavior</i> y prototipar todos los posibles. | 100 % | |
| 1.4. Sistema de debug visual: vectores. | 100 % | |
| 1.5. Herramientas control de ejecución. | 50 % | It.7 |
| 2.1. Trabajar más la memoria. | 65 % | It.9 |
| 2.2. Terminar de definir idea (GDD). | 100 % | |

| Iteraciones | % Completado | Terminado en |
|--|--------------|--------------|
| 2.3. Implementar el mínimo producto. | 20 % | It.4 |
| 2.4. Experimentación y análisis de resultados. | 0 % | It.8 |

Tabla 5.2: Resumen fase 0

Esta primera fase engloba las tres primeras iteraciones donde comenzamos a terminar de definir la idea para el proyecto, ya que, todavía era un poco difusa la imagen del producto final que se quería desarrollar.

Iteración 0: Bucle del juego y primeros sistemas

Para ir entrando en una dinámica productiva iniciamos el desarrollo de funcionalidades básicas como pueden ser el bucle principal del juego, en el cual usamos de la librería `<chrono>` de C++ para medir y limitar el tiempo entre ejecuciones del bucle acompañado del uso de una enumeración que contiene los posibles resultados de un ciclo del juego.

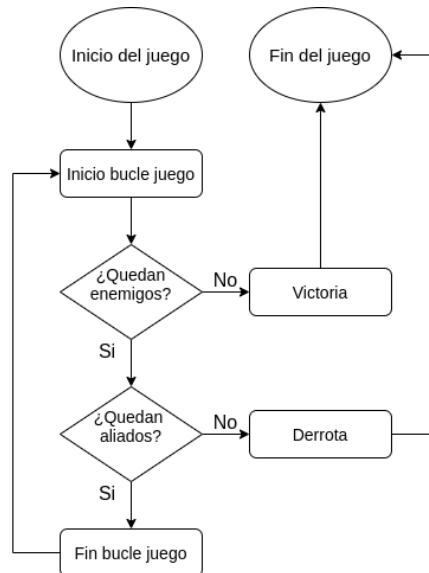


Figura 5.1: Posibles resultados del bucle de juego.

Una vez tenemos el bucle, toca introducir algo que hacer en él, como primer sistema del juego encontramos el sistema de dibujado el cual se encargará de crear y manejar la ventana, además de pintar nuestras entidades y herramientas visuales. Para incluir todas estas funcionalidades sin añadir librerías pesadas incluiremos `<TinyPTC>` de

Francisco J. Gallego-Durán and Glenn Fiedler and Fred Howell and Alessandro Gatti, la cual no incluye demasiadas funcionalidades u opciones pero nos permitirá trabajar rápido y de forma sencilla. Los sprites en un inicio serán simples cuadrados de colores por lo que no requeriremos mucho.

Por último, se añadieron las componentes de movimiento e IA que, junto a sus correspondientes sistemas, nos permitirán poder crear un comportamiento de patrulla básico entre puntos fijos. Además, se terminó la preparación de las herramientas de L^AT_EXy ‘Jabref’ para redactar la memoria acompañado de la lectura de las directrices, normas y memorias de compañeros del año pasado.

Iteración 1: Entero con coma fija e inicio de la IA

A lo largo de la iteración siguiente, seguimos añadiendo funcionalidades al prototipo como puede ser el componente y sistema de *Input*, el cual nos permitirá interactuar con el juego mediante el mapeado del teclado haciendo uso de la librería `<X11>` y `<TinyPTC>`.

Una de las herramientas implementadas en el proyecto es el tipo de dato entero con coma fija, en C++ podemos encontrar el tipo fundamental *float* creado para trabajar con números reales, pero si miramos cómo funcionan exactamente es posible encontrar que no es lo que deseamos exactamente. Los *floats* usan “precisión simple” lo cual implica que conforme el número sea más grande, la cantidad de bytes destinados a la representación de la parte decimal va disminuyendo ofreciendo así menor precisión. Además, trabajar con ellos es más costoso a nivel computacional¹ y llevan un código más pesado debido a la codificación usada al ser compilados.

El tipo con coma fija tiene dos partes, en primer lugar el valor que se quiere representar normal y corriente, y por otro la escala, número que indicará la cantidad de valores disponibles para la representación decimal. La escala la usaremos para multiplicar o dividir nuestro número para convertirlo a coma fija, devolverlo sin escalar y/o operaciones con otros números enteros.

Es importante elegir una escala adecuada para trabajar de forma eficiente. Para ello escogeremos un número portencia de 2 el cual nos permitirá usar desplazamientos en multiplicaciones y divisiones, haciendo los cálculos mucho más rápidos que si tuvieramos que usar las operaciones comunes.

Por otro lado, se comenzó a trabajar en la IA del juego creando los primeros comportamientos y herramientas para alternar entre ellos. Aquí caeremos en los primeros errores de concepto a la hora de trabajar con los ‘Steering behaviors’, ya que, a la hora de implementar los usamos un plateamiento y cálculos de movimiento cinético sin aceleración, trabajando directamente con la velocidad.

¹Esto pasa sobretodo en máquinas que no tengan CPUs con unidades destinadas a cálculos en coma flotante

| | |
|--|--|
| 64 bits = 48 bits parte entera + 16 bits parte decimal | |
| Escala = $65536 = 2^{16}$ | |
| Conversión a coma fija | Conversión a decimal |
| $2.5 \times 65536 = 163840$ | $\begin{array}{r} 163840 \\ 65536 \\ \hline 327680 & 2.5 \\ 0 \end{array}$ |
| Multiplicación entera | |
| $5 \times 15 = 75$ | |
| Multiplicación coma fija | |
| $327680 \times 983040 / 65536 = 4915200$ | |

Figura 5.2: Ejemplos entero con coma fija.

En su lugar, lo planteado por la técnica es usar la aceleración de la entidad, de esta forma se logrará calcular el movimiento deseado en un momento puntual, enfrentando lo contra la velocidad acumulada. Esto nos proporcionará un movimiento continuado y dinámico en lugar de cambiar drásticamente de dirección y velocidad.

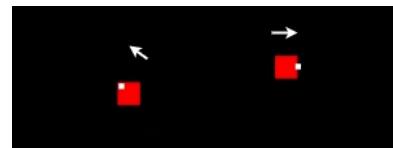


Figura 5.3: Movimiento rectilíneo uniforme.



Figura 5.4: Movimiento rectilíneo acelerado.

Uno de los comportamientos más sencillos es el ‘Arrive’, el cual nos llevará a la posición objetivo de forma directa y conforme estemos llegando a esta, la aceleración disminuirá parando una vez hayamos llegado al destino. En esta función podremos apreciar cómo calculamos la velocidad objetivo en función de la distancia hasta llegar a la posición objetivo, una vez se calcula se compara con el desplazamiento actual de la entidad y es la resta de ambas la que nos dará la aceleración necesaria para alcanzar la velocidad objetivo, siempre controlando que la aceleración y la velocidad no exceden los límites marcados.

Además, contamos con dos modificadores adicionales como son el tiempo deseado para alcanzar el objetivo y un límite de distancia con el objetivo para evitar solapes y/o oscilar sobre él.

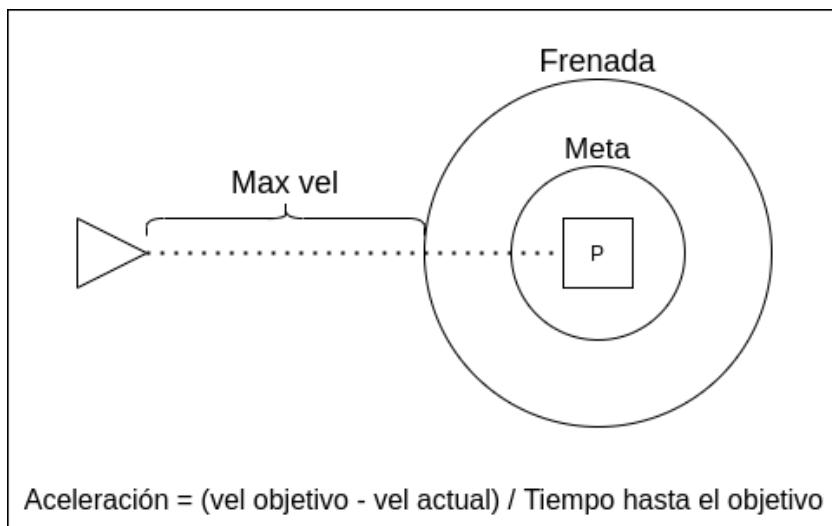


Figura 5.5: Diagrama Arrive Behaviour.

Iteración 2: Herramientas de depuración visual y tiempo de ejecución

Durante la tercera iteración nos centramos en la mejora de algunos '*Steering behaviors*', para terminar de hacerlos más fáciles de usar y combinar, con el fin de tener un código más limpio y legible, también se mejoró el uso de la coma fija añadiendo más operaciones y ajustando el funcionamiento.

En cuanto al '*Flocking*', se comenzó a trabajar en el cálculo de las distintas fuerzas que actuarán sobre las unidades, siendo estas: la atracción al centro de masas del grupo y la separación de las unidades cercanas. En ambos casos, se recorre el array de entidades aliadas seleccionando las que se encuentren dentro del radio de influencia, el módulo del vector dependerá de la distancia entre ambas entidades siendo las más cercanas la que tengan más presencia en la separación y las más lejanas en la cohesión, una vez se terminan de acumular todas las direcciones se comprueba que la fuerza resultante no es superior al límite.

Por otro lado, se añadieron herramientas para debug del movimientos de las entidades y las diferentes componentes de este, para el dibujado de los vectores se eligió el algoritmo de '*Bresenham*', ya que, nos permite dibujar rectas de una forma efectiva y sin consumir muchos recursos. Además, el algoritmo esta diseñado para trabajar con enteros, cosa que hacemos a lo largo de todo el prototipo. En el esquema 5.7 la cuadricula blanca representa los pixeles de la pantalla, en color azul la recta ideal que se busca dibujar y

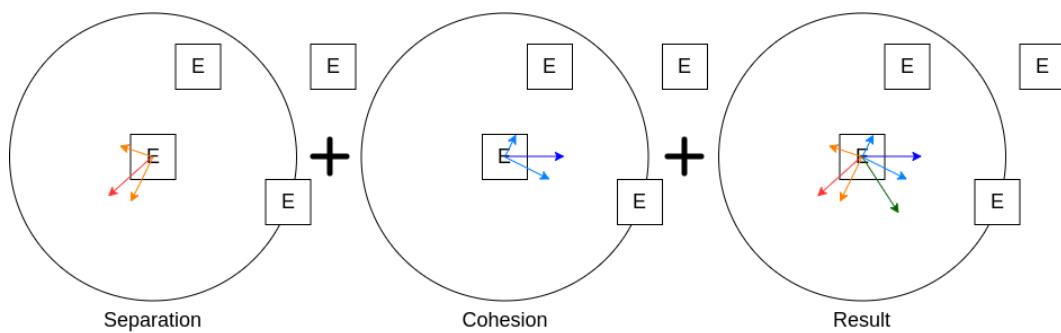


Figura 5.6: Esquema fuerzas del flocking

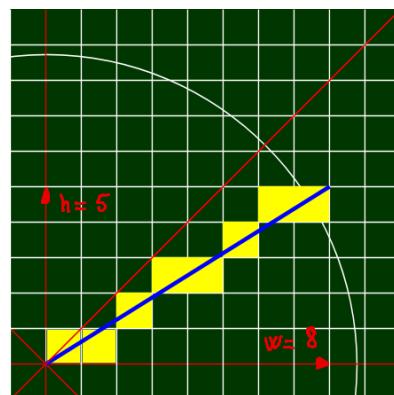


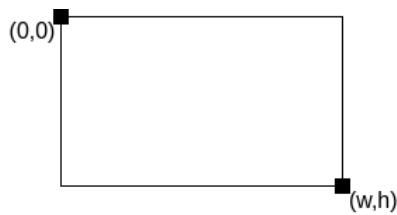
Figura 5.7: Esquema algoritmo de Bresenham.

en amarillo la recta resultante del algoritmo.

Otra de las herramientas destinadas a poder analizar el funcionamiento del programa es el control del tiempo de ejecución y el tamaño del ‘*DeltaTime*’. Al poder disminuir el número de fotogramas por segundo, podemos visualizar más detenidamente qué sucede en pantalla y ver que todo va según la previsión. Por otra parte, se puede aumentar la velocidad para llegar antes a un punto de la ejecución en concreto. Además, jugar con el tamaño del ‘*DeltaTime*’ nos permite experimentar con los valores de velocidad y/o otros factores interpolados para ajustarlos de forma visual.

Por último en cuanto a implementación, toda la información relacionada con el sistema de dibujado y los elementos visuales del juego se trabajan con *uint64_t*, mientras que todo lo relacionado con el movimiento y demás cálculos físicos se trabajan con *float_t<int64_t>*. Por ello es necesario usar “*casts*” sobre los datos físicos con los que se quiera operar en el *Render System*², como es una operación recurrente en el código se implementó un sistema auxiliar para convertir coordenadas continuas (usadas en la mayoría de sistemas) a coordenadas de pantalla o pixel.

²Ya sea la posición en el mapa como el sistema de debug para los vectores de *Steer*.



Coordenadas de pantalla.



Coordenadas continuas.

En lo referente a la memoria comenzamos la redacción de una versión preliminar del Game Design Document (GDD) donde comenzamos a describir las características del producto, del Estado del Arte donde se hace mención a juegos que nos han servido de inspiración o modelo para imitar y se incluye una explicación sobre las técnicas y algoritmos se van a usar, y la explicación sobre la metodología seguida durante este desarrollo.

5.2. Fase 1: Producto mínimo viable

| Iteraciones | % Completado | Terminado en |
|---|--------------|--------------|
| 3.1. Cambios en el motor ECS. | 100 % | |
| 3.2. Cambios en el almacenamiento de datos. | 100 % | |
| 3.3. Cambio tipos propios a template. | 100 % | |
| 3.4. Cambio en el sistema de clases. | 100 % | |
| 4.1. Añadir referente que use <i>Steering Behavior</i> . | 100 % | |
| 4.2. Diagrama y explicación de las fases de una iteración. | 100 % | |
| 4.3. Agrupar iteración en fases y describir el desarrollo hasta la fecha. | 30 % | It.5 |
| 4.4. Implementar mínimo producto. | 85 % | It.6 |
| 4.5. Análisis de resultados del MVP. | 10 % | It.7 |
| 5.1. Crear unidad de ataque a distancia. | 100 % | |
| 5.2. Adición de formación en anillo. | 80 % | It.6 |
| 5.3. Balanceo de las unidades. | 0 % | It.8 |
| 5.4. Profundizar los objetivos en el GDD y explicar método de balanceo. | 0 % | It.8 |
| 5.5. Incluir tabla resumen al inicio de fase e incluir información extra como el set-up. | 100 % | |
| 6.1. Experimentar con la formación en anillo y fuerzas cuadráticas. | 100 % | |
| 6.2. Sistema de proyectiles para terminar la unidad a distancia. | 100 % | |
| 6.3. Reemplazar TinyPTC por Dear IMGUI y hacer una interfaz básica. | 100 % | |
| 6.4. Ajustar y continuar con la memoria. | 80 % | It.8 |
| 6.5. Profundizar en el GDD y listar cuestiones tecnológicas que hacen falta. | 100 % | |

Tabla 5.3: Resumen fase 1

Iteración 3: Motor ECS y gestión de memoria

La fase comenzó con una iteración corta debido a las vacaciones de Navidad y final de año, en la que el foco estuvo en refactorizar y dejar en mejor estado el código desarrollado hasta el momento, donde se separó el núcleo principal del motor ECS del conjunto de sistemas y herramientas en dos *namespaces* diferentes.

Por el lado del motor ECS, en un primer lugar el almacén de componentes estaba diseñado para contener punteros a componentes. Esto implicaba que los datos de los componentes se guardaban en la memoria libre de forma “anárquica” 5.8, en su lugar, lo deseado es almacenar todas las componentes de forma contigua para poder utilizar la caché cuando sea posible, para poder aprovechar que es más rápida y minimizar la cantidad de veces que mandamos al sistema ir a buscar nuestros datos 5.9.

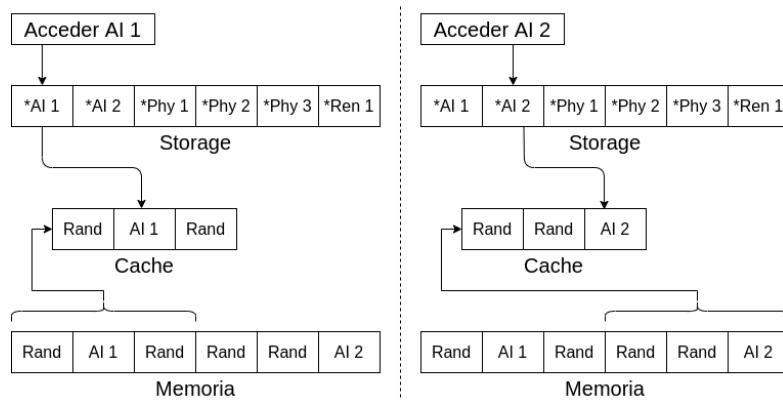


Figura 5.8: Esquema acceso a vector de punteros.

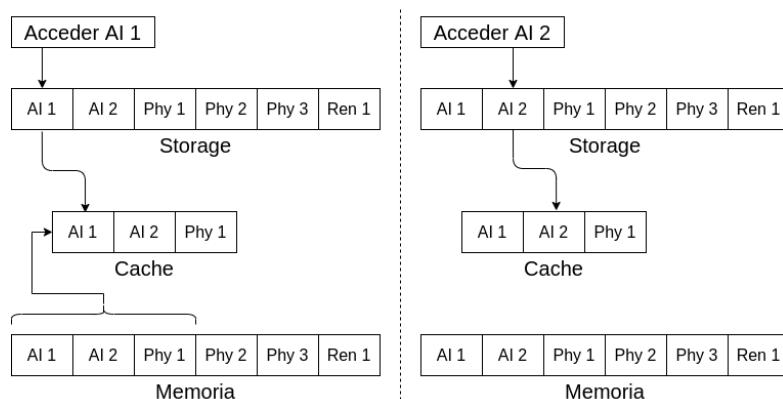


Figura 5.9: Esquema acceso a vector de objetos.

A continuación, se modificó la estructura de las entidades para componerse únicamente de su *Entity ID* y una serie de *Component IDs* para posibles comprobaciones de que

componentes contiene la entidad, dejando así de tener un puntero a las componentes del almacén. Esto implica una serie de cambios en la fachada y uso del *Entity manager* a la hora de crear/eliminar entidades y la obtención de las componentes desde los sistemas a través del contexto.

Por último se siguieron modificando los tipos propios, tanto el *fint_t* como los *vec2* y *fvec2* para trabajar como plantillas, pudiendo así librarnos de la necesidad de crear un tipo completo para cada tipo básico con el que queramos usarlos.

Iteración 4: Combate, victoria y derrota

Durante el mes de enero se implementaron diversas funcionalidades, en primer lugar abordamos el sistema de combate del juego. Para ello se creó una componente para almacenar elementos como la vida de la unidad, su daño, el rango y el tiempo entre ataques. Además, los sistemas de ataque para ir restando vida a las unidades conforme reciban daño, el de *cooldown* para actualizar los temporizadores para volver a atacar, y por último el sistema de muertes donde se manda a pedir que se borren las entidades que hayan muerto y se realizan las comprobaciones de victoria y derrota de la partida.

Una vez se resuelve la partida, en el bucle de ejecución del juego se reiniciarán el nivel vaciando y volviendo a cargar los elementos del juego.

Para incluir los nuevos sistemas y funcionalidades a la rutina de las unidades, se ha ampliado el apartado de toma de decisión incluyendo métodos para detectar si hay unidades enemigas cerca y seleccionar como objetivo a la más cercana, que en caso de morir se lanzará una nueva búsqueda. De forma similar, cuando no haya unidades enemigas cercanas se regresará al estado de patrulla (en caso de la IA) y las manejadas por el jugador volverán a la posición de su marcador en el mapa.

En cuanto a la gestión de las unidades en partida, se creó un manejador de equipos donde se gestiona que unidades pertenecen a cada facción, y proporcionar métodos para obtener los *IDs* de las unidades por equipo e ir actualizando los equipos cuando haya bajas. Además de, tener métodos para crear tipos de unidades en concreto trabajando a la vez de “fábrica”.

Por otro lado, en el apartado visual, se ha añadido un marcador que indica la dirección a la que se está orientando la entidad con el fin de dar *feedback* al jugador y que se entienda mejor y de forma visual el desplazamiento o acción actual. Este marcador tiene un tamaño relativo al del sprite de la unidad y serán siempre de color blanco para que sea fácil de identificar.

Por último, en cuanto a la memoria, se ha añadido en el apartado de metodología la descripción de las fases que componen una iteración junto a un esquema. Se ha introducido un tercer referente el cual basa toda su jugabilidad alrededor de los *Steering behavior*, debido a este juego hemos optado por cambiar la jugabilidad del proyecto con



Sin marcador.

Con marcador.

Figura 5.10: Marcador dirección entidad.

el fin de imitar la de este juego y con intención de innovar en los controles, como ya se refleja en el GDD, actualmente el jugador no maneja directamente a las unidades con el ratón ni las selecciona de forma individual, sino que controla un marcador verde el cual será seguido por sus tropas y mediante ordenenes activará los distintos comportamientos de sus unidades.

Iteración 5: Mecánicas fallidas

A lo largo de la siguiente iteración se trabajó sobretodo en las mecánicas de juego, con el fin de acercarse un poco más a la obtención de un prototipo que cumpla con los requisitos del MVP. En concreto se comenzó el desarrollo de un segundo tipo de unidad basada en ataques a distancia y la posibilidad de controlar la formación que siguen nuestras unidades, pudiendo alternar entre un movimiento sin orden o una formación en anillo.

En cuanto a la unidad a distancia, gracias a que tenemos la componente de combate podemos ajustar de una forma muy sencilla el rango de ataque de cada unidad, dejándonos la creación de un sistema de proyectiles como tarea para completar el funcionamiento de dicha unidad.

Por otro lado encontramos la formación en anillo, la cual cuenta con una primera versión que se basa en modificar el rango de cohesión al centro del grupo en función de la distancia de ataque. Probando esta forma de configurar la posición de las unidades nos hemos dado cuenta de que no muestras buenos resultados en conjuntos grandes de unidades, ya que, numerosas unidades quedan atrapadas en posiciones incorrectas y la distribución resultante no es muy óptima.

Por último, se ha introducido en la memoria una lista con las herramientas que han sido utilizadas durante el desarrollo, donde se incluye el nombre de las librerías, la versión y los sistemas sobre los que puede ser lanzado el juego. Todo con el fin de ayudar a quién

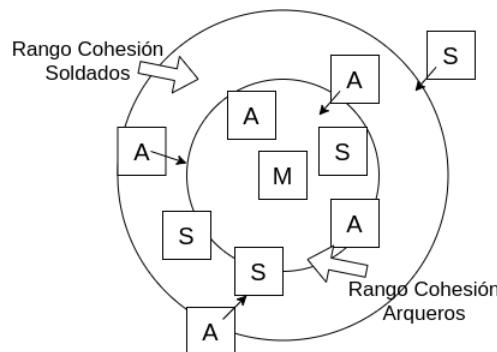


Figura 5.11: Formación anillo por rango de cohesión

quiera utilizar, recompilar y/o modificar el prototipo. Además, se ha añadido al comienzo de cada fase una tabla resumen donde se indican las tareas principales de cada iteración, el portentaje estimado de completado y, en caso de no haber sido terminada, la iteración donde se ha finalizado.

Iteración 6: Interfaz y Dear ImGui

En la cuarta y última iteración de esta fase, el objetivo es el de terminar todas las mecánicas del prototipo y sustituir *<TinyPTC>* por *<Dear ImGui>* para poder centrarnos más adelante en el diseño del nivel/es del prototipo de cara a la entrega final. Además de, poder añadir alguna funcionalidad extra si es necesario para crear una experiencia de juego mejor.

Llegados a este punto, queremos crear una interfaz y tener la posibilidad de poder añadir algún elemento visual al proyecto, por lo que necesitamos de una librería que ofrezca más opciones a la hora de trabajar con ella. En este caso hemos elegido *<Dear ImGui>* de Omar 'ocornut', la cual, propone una nueva forma de trabajar las interfaces³ en auge actualmente y además, nos proporciona una serie de *widgets* y herramientas pre-definidas que nos permitirán crear un HUD fácil y sencillo. Además, funciona con *GLFW* y *OpenGL* así que si necesitamos revisar el funcionamiento de alguna funcionalidad, no tendremos excesivos problemas ya que hemos trabajado con ellas anteriormente.

Para comenzar a trastear con esta librería, hemos implementado una pequeña ventana que nos permitirá manejar en tiempo real el número de veces que se ejecutará el bucle del juego por segundo y/o el tamaño del *DeltaTime*, además tiene un *checkbox* para activar el debug visual del movimiento de las entidades.

Por otro lado, se ha creado una segunda ventana que hará de "minimapaz" mostrará la ubicación de los unidades tanto aliadas como enemigas y el marcador. Esto nos puede

³ *IMGUI (Immediate Mode GUI) paradigm.*

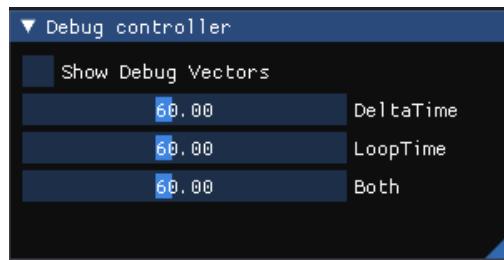


Figura 5.12: Herramienta depuración en tiempo de ejecución

servir de cara a una versión del juego en el que el mapeado sea más grande que la pantalla, para que nos permita visualizar y ubicarnos en el nivel.

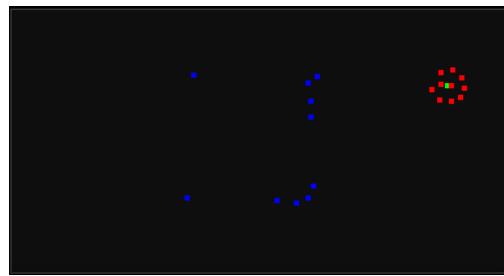


Figura 5.13: Mini Mapa.

En cuanto a las mecánicas del juego, hemos añadido un sistema de lanzamiento de proyectiles para las unidades que ataquen a distancia. Esto incluye: la creación de las balas siguiendo la orientación del personaje, la destrucción de los proyectiles una vez alcanzan su distancia máxima y/o impactan contra un objetivo. La creación y destrucción de entidades se realiza al final del bucle del juego, al igual que con las demás entidades.

Un proyectil consiste en una entidad formada por una componente física para su desplazamiento, una para el cálculo de colisiones, y una visual para poder dibujarla. Una vez se lance un proyectil, este seguirá la dirección que tenía la unidad al momento de disparar, por lo que solo hará daño si la unidad enemiga no cambia de posición lo suficiente para esquivar el área de impacto. A la hora de programar las colisiones, solamente contaremos las unidades del bando contrario para optimizar el número de comprobaciones, para ello tendremos un vector de proyectiles por cada bando.

Otra de las mecánicas que han sido mejoradas es la formación en anillo de nuestras unidades. Los resultados y la forma generada eran poco satisfactorios por lo que se cambió de trabajar con la intensidad de la cohesión a calcular una posición deseada para cada unidad en relación a su distancia de ataque. Para ello, mediante el trazado de rayos creamos una recta entre el marcador del jugador y cada una de las distintas unidades, sobre esa recta se marcará un punto el cual deberá ser alcanzado por la unidad, de esta forma solventamos los solapamientos entre unidades y que se queden encerradas en lugares

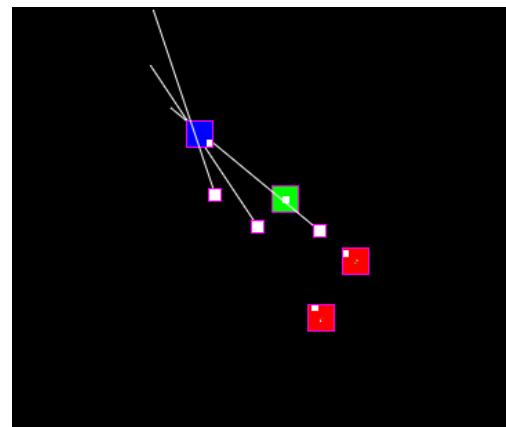


Figura 5.14: Demostración sistema de disparos.

incorrectos, además de ser más escalable en caso de adición de nuevas unidades en el futuro.

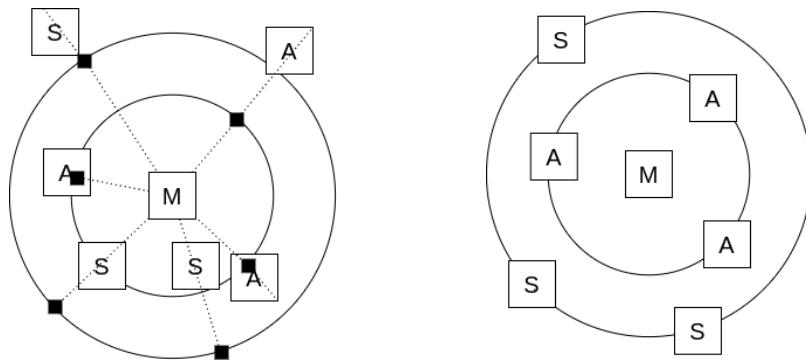


Figura 5.15: Formación anillo con trazado de rayos.

Por último se ha cambiado el cálculo de las fuerzas de cohesión y separación para que tenga un crecimiento cuadrático en lugar de lineal, de esta forma reducimos su presencia en la media/larga distancia y la aumentamos en las cercanías.

| Distancia | Lineal | Cuadrado |
|-----------|--------|----------|
| 40 | 5.59 | 31.25 |
| 60 | 2.484 | 6.1728 |
| 80 | 1.3975 | 1.953 |
| 100 | 0.8944 | 0.8 |
| 120 | 0.621 | 0.3858 |
| 140 | 0.4563 | 0.2082 |

| Distancia | Lineal | Cuadrado |
|-----------|--------|----------|
| 160 | 0.3494 | 0.1221 |
| 180 | 0.2760 | 0.0762 |
| 200 | 0.2236 | 0.05 |
| 220 | 0.1848 | 0.0342 |

Tabla 5.4: Fuerzas de separación en función de la distancia

5.3. Fase 2: Últimos pasos

| Iteraciones | % Completado | Terminado en |
|--|--------------|--------------|
| 7.1. Continuar trabajando el GDD. | 10 % | It.9 |
| 7.2. Ajustes de valores de daño y vida. | 0 % | It.8 |
| 7.3. Incluir los ciclos de análisis de los comportamientos resultantes y propuestas de mejora | 40 % | It.8 |
| 7.4. Revisar bugs y comportamientos erróneos. | 100 % | |
| 7.5. Incluir sistema de cámara y coordenadas de mundo. | 100 % | |
| 7.6. Implementar jugabilidad básica: debe poder jugarse una partida simple al juego, con las reglas propuestas. | 70 % | It.8 |
| 8.1. Balanceo de unidades y ajustar sus valores. | 100 % | |
| 8.2. Redactar de forma detallada el balanceo. | 100 % | |
| 8.3. Prueba de compilación cruzada para Windows. | 0 % | It.9 |
| 8.4. Describir todas las herramientas implementadas y procesos de análisis. | 100 % | |
| 9.1. Compilación cruzada para Windows. | 100 % | |
| 9.2. Finalizar la memoria | 100 % | |
| 9.3. Preparar la entrega | 100 % | |

Tabla 5.5: Resumen fase 3

Iteración 7: Terminando las cuestiones tecnológicas

A lo largo de esta iteración los objetivos propuestos eran los de terminar de implementar y/o corregir las mecánicas y sistemas del proyecto, a la vez que se continua en la redacción de la memoria y el GDD. De esta forma poder dejar las ultimas iteraciones en diseñar los niveles, balancear el juego y terminar de redactar la memoria.

Uno de los puntos principales cae sobre la resolución de un error en las colisiones de las balas. Este problema consistía en una serie de balas que atravesaban a las unidades

con baja frecuencia y sin motivo aparente. Con el fin de subsanar este problema se ha creado una componente de colisiones que ha sustituido el uso del tamaño del sprite en los cálculos de colisiones con el límite de la ventana y demás entidades.

Al integrar esta componente en el sistema de debug visual pudimos observar como el problema era que las colisiones estaban siendo calculadas teniendo en cuenta el centro de la entidad, cuando el origen de ésta es la esquina superior izquierda, esto suponía un desplazamiento de todas las colisiones y por lo tanto errores en las detección, con cambiar ese factor se solucionaron todos los problemas.

A lo largo de la ejecución del programa, podíamos ver cómo algunas unidades se quedaban quietas sin poder retomar su movimiento hasta que otra unidad la desplazaba y hacía reaccionar. Después de analizar la situación pudimos determinar que el problema ocurría cuando la unidad perdía su velocidad y aceleración dentro del rango de “velocidad reducida” del ‘Arrive behaviour’. Lo que estaba sucediendo es que no se estaban calculando correctamente las divisiones con enteros de coma fija cuyo resultado era un valor entre 0 y 1.

$$\boxed{(\text{Fijo_1} / \text{Fijo_2}) * \text{Escala} = \text{Resultado}}$$

$$(2293760 / 2621440) * 65536$$

$$0 * 65536 = 0$$

$$\boxed{(\text{Fijo_1} * \text{Escala}) / \text{Fijo_2} = \text{Resultado}}$$

$$(2293760 * 65536) / 2621440$$

$$150323855360 / 2621440 =$$

$$57344 = 0,875$$

Figura 5.16: Comparación resultados división.

En una primera versión del operador, el cálculo que se realizaba era: dividir primero los valores a operar y después multiplicar el resultado por la ‘Escala’. Este orden de los factores nos llevaba a que si, en la primera operación el divisor era mayor que el dividendo, el valor del resto era 0, ya que trabajamos con enteros y trunca los valores decimales.

Para solucionar el problema, lo que se hizo fue extraer una versión reducida del tipo de dato que sólo contaba con las operaciones de división e introducirla en ‘Godbolt’⁴, para poder realizar una serie de pruebas y ajustes rápidamente sin tener que estar pendiente del resto del proyecto.

Una vez teníamos ya las funciones listas, lo único que quedaba era experimentar con el orden de los factores a la hora de realizar la operación y ver si el resultado variaba. Tras un ligero ajuste conseguimos que la operación conservara el valor deseado, esto se logró multiplicando el primer valor de la operación por la ‘Escala’ antes de dividir por el

⁴<https://godbolt.org>

segundo dígito, de esta forma evitamos los resultados entre 0 y 65536 y que se trunque el valor en mitad de la operación.

Una vez solventados los errores, nos pusimos a crear lo necesario para poder diseñar niveles más amplios. Para ello, incluimos un sistema de cámaras, que mostrará la porción del nivel donde se encuentre nuestro puntero y unidades. El seguimiento se hace utilizando el ‘*Arrive behaviour*’ de forma que el movimiento de la cámara comienza cuando nos sepáramos una cierta distancia del centro de la escena.

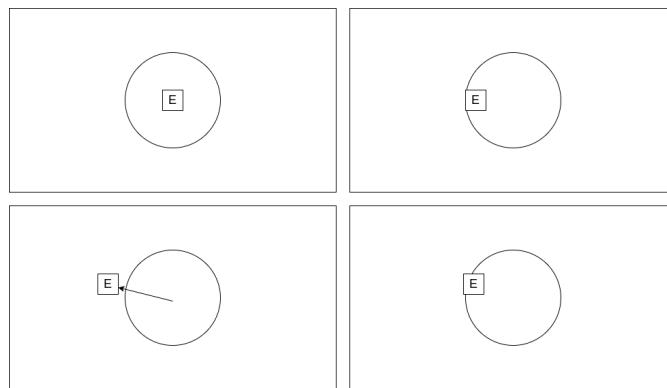


Figura 5.17: Arrive behaviour en la cámara

Ahora las dimensiones de la pantalla pasan a ser las de la cámara y el nivel tendrá unas dimensiones propias, las cuales usaremos para limitar el movimiento de las unidades, por lo que las unidades se pueden mover libremente fuera de nuestro campo de visión siempre y cuando estén en los límites del nivel. Mientras estén fuera de cámara solo se dibujarán en el minimapa para evitar hacer llamadas inútiles.

Además de los cálculos para las colisiones, otro requisito fue adaptar el minimapa para que mostrara qué parte del nivel está siendo visualizado y que nos permita ubicar a todas las unidades y objetivos fuera de cámara. Un detalle importante es que al crear el nivel, le indicamos al motor la relación entre las dimensiones de pantalla y nivel, por lo que los objetos mostrados en el minimapa se ajustarán en función de este parámetro: cuánto más grande el nivel, más pequeños serán los elementos.

En lo referente al *HUD*, se ha añadido también un selector de comportamiento y formación para nuestras unidades, de esta forma podremos controlar dichos parámetros con simples *clicks* y ahorrarnos el tener que disponer de una tecla para cada opción.

En lo referente a la mecánicas y comportamientos de las entidades ha habido dos cambios relevantes durante la iteración: en primer lugar, encontramos la adición de un componente *singleton* de *Blackboard* que nos permitirá almacenar información compartida para todas las unidades bajo nuestro control, además de controlar la frecuencia en la que la esta información se actualiza, lo que nos permitirá crear una sensación de retraso en la reacción a nuestras acciones por parte de las unidades a la vez que nos permite eliminar la necesidad de guardar la misma información en cada componente de IA por

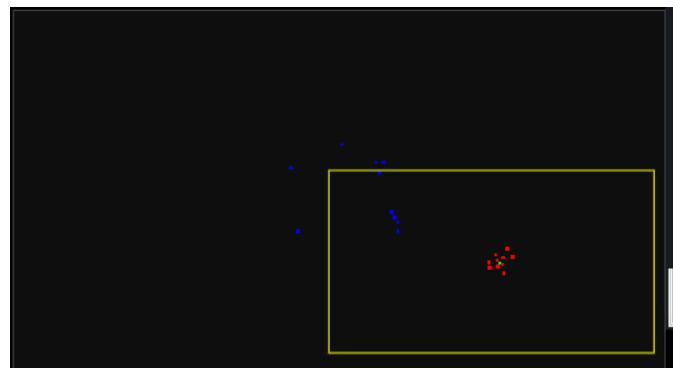


Figura 5.18: Cambios en el minimapa



Figura 5.19: Selector de comportamiento y formación

unidad que tengamos.

En segundo lugar encontramos otro cambio en las componentes de IA, esta vez con efectos sobre las unidades controladas por la máquina. En este caso se trata de la creación de una estructura auxiliar que representa y contiene las rutas que las unidades seguirán mientras no estén peleando contra nosotros.

En un primer momento estas rutas se encontraban directamente en la componente. Actualmente en la componente podemos encontrar un “iterador de ruta” el cual nos podrá facilitar la siguiente posición y gestionar como avanzamos por la patrulla, de manera que si llegamos al final de forma automática nos devolverá la primera posición que recorrimos. De este modo evitamos errores en la gestión de las patrullas y dublicar la información de estas.

Iteración 8: Diseño de niveles

Durante esta penúltima iteración buscamos conseguir un prototipo que ofrezca una experiencia de juego capaz de mostrar todas las herramientas diseñadas y entretenner al jugador. Para ello se han planteado objetivos relacionados con el diseño de los niveles y el ajuste de las unidades, tratando de conseguir el que el jugador sepa en todo momento su objetivo en el juego y que sea atractivo para el.

En primer lugar, hemos trabajado el equilibrio de los valores de vida y daño de los distintos tipos de unidades. Partiendo de una situación con todas las unidades con los mismos valores debemos tratar de darles ventajas y desventajas a cada una de ellas, con el fin de obligar al jugador a pensar bien cómo colocarlas y utilizarlas. Durante el proceso se ha desarrollado un excel para observar los valores y modificarlos de forma rápida, esto nos ayuda a poder hacer un análisis profundo de los efectos de cada valor en el resultado de las unidades.

| Tiempo de refresco | Ataque | Daño por segundo | Vida enemiga | Tiempo para matar | Impactos para matar |
|--------------------|--------|------------------|--------------|-------------------|---------------------|
| 2 | 4 | 2 | 12 | 6 | 3 |
| 2 | 4 | 2 | 14 | 7 | 3.5 |
| 2 | 5 | 2.5 | 12 | 4.8 | 2.4 |
| 2 | 5 | 2.5 | 14 | 5.6 | 2.8 |
| 3 | 5 | 1.667 | 12 | 7.2 | 2.4 |
| 3 | 5 | 1.667 | 14 | 8.4 | 2.8 |

Tabla 5.6: Valores arquero vs soldado

La idea es lograr que el jugador sienta que los Soldados son unidades más resistentes pero con menor daño. Para ello les dotaremos de mayor vida pero menor daño que a los arqueros. Por otro lado, realizar un movimiento con las manos es una acción que requiere mucha menos preparación y dedicación que cargar un arco, por ello se contempla darles una mayor frecuencia a sus ataques.

| Tiempo de refresco | Ataque | Daño por segundo | Vida enemiga | Tiempo para matar | Impactos para matar |
|--------------------|--------|------------------|--------------|-------------------|---------------------|
| 2 | 4 | 2 | 6 | 3 | 1.5 |
| 2 | 4 | 2 | 8 | 4 | 2 |
| 2 | 5 | 2.5 | 6 | 2.4 | 1.2 |
| 2 | 5 | 2.5 | 8 | 3.2 | 1.6 |
| 3 | 5 | 1.667 | 6 | 3.6 | 1.2 |
| 3 | 5 | 1.667 | 8 | 4.8 | 1.6 |

| Tiempo de refresco | Ataque | Daño por segundo | Vida enemiga | Tiempo para matar | Impactos para matar |
|--------------------|--------|------------------|--------------|-------------------|---------------------|
|--------------------|--------|------------------|--------------|-------------------|---------------------|

Tabla 5.7: Valores soldado vs arquero

Los arqueros, por su parte, están pensados como unidades de mediano alcance que atacan con proyectiles. Son débiles y si tratan de avanzar solos podrían ser masacrados por grupos de soldados. Para evitarlo, el jugador debería intentar proteger a sus arqueros utilizando a sus guerreros como muro defensivo. A pesar de que realizan más daño a sus rivales, los proyectiles no realizan un daño inmediato, sino que actúan al impactar, cosa que podría no suceder. Esto nos ayuda a equilibrar la balanza debido a la ventaja de rango que poseen los arqueros.

En segundo lugar encontramos el diseño de los niveles, a la hora de establecer la cantidad de enemigos y su disposición tenemos que tener en cuenta el objetivo del nivel, si el nivel consiste en limpiar todos los enemigos existentes tendremos que evaluar la cantidad y cómo se agrupan para que puedan ser limpiados si gestionas medianamente bien las unidades. Si el objetivo es llegar a un punto en concreto del nivel, los enemigos serán más y el combate directo nos pondrá en aprietos, por lo que el objetivo debería ser encontrar una ruta que nos asegure el menor nivel de conflicto posible.

Iteración 9: Retoque y preparación entrega

Durante esta última iteración, debido al tiempo y a el estado del proyecto, lo único que queda por hacer es desarrollar dos pequeñas herramientas para el proyecto, la compilación en *Windows* y preparar todos los documentos para la entrega.

En este último período se ha creado un pequeño sistema de *TiggerBoxes* que nos permiten lanzar eventos del juego una vez colisionamos con ellas con nuestro marcador, esto nos habilita poder tener niveles en los que podremos avanzar simplemente alcanzando la localización objetivo. Estos objetos se componen únicamente de una componente de colisiones, una de movimiento una de dibujado y un nuevo componente de eventos que contiene la acción a ejecutar cuando colisionemos , los textos salen representados en blanco en el minimapa.

Además, ahora contamos con la posibilidad de ubicar textos estáticos por el nivel para dar información al jugador cuando lo veámos necesario, esto nos permitirá darle píldoras de información como los controles iniciales o el objetivo del nivel en el que se encuentra, además de, introducir elementos nuevos que se introduzcan en los niveles posteriores.

En cuanto a la compilación y ejecución en *Windows*, con un par de cambios en las opciones de compilación y añadiendo las “*.dll*” necesarias el proyecto funciona correctamente y fluido en el sistema. Para la compilación de *Dear ImGUI* lo único que ha sido necesa-

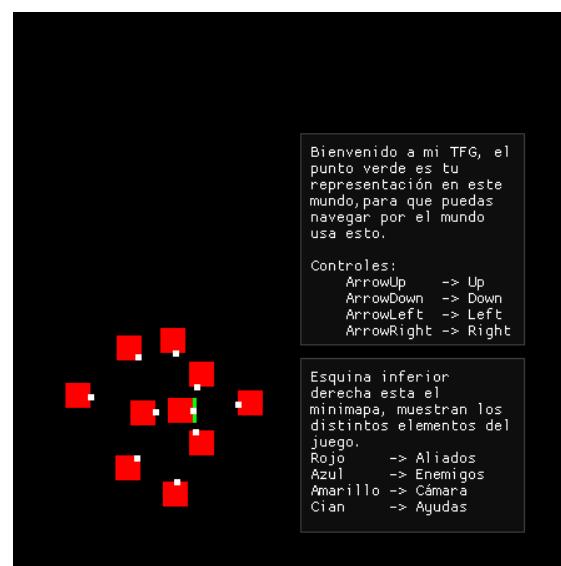


Figura 5.20: Mensajes estáticos nivel

rio añadir ha sido la opción de compilación “*-DIMGUI_DISABLE_WIN32_FUNCTIONS*” para evitar que cargue funciones de *DirectX11*. El compilador usado ha sido *MinGW64* a través del entorno *MSYS2* corriendo directamente en *Windows*.⁵

Para finalizar, lo último que queda es terminar de preparar la entrega y todos los documentos relacionados como son: finalizar la memoria con la redacción de las conclusiones, realizar el resumen del trabajo, diseñar un poster para la presentación, adjuntar el documento de autoría e incluir una licencia para el proyecto.

Una vez este todo se creará un entregable con todos los contenidos dispuestos de la mejor forma posible para el tribunal y se entregará de acuerdo con la fecha establecida.

⁵Página de MSYS2: y guía para usar MinGW <https://www.msys2.org>

6 Conclusiones

Para hablar de las conclusiones obtenidas del desarrollo del proyecto, quiero traer de vuelta los objetivos que se propusieron en su momento.

Uno de los objetivos más generales era “Comprender mejor el funcionamiento del PC”, gracias a tareas como el desarrollo del tipo numérico en coma fija hemos ampliado nuestros conocimientos sobre como traduce y codifica el código el compilador.

Además, tras realizar de nuevo el proceso de preparar el proyecto para no depender del sistema para su ejecución, hemos afianzado nuestros conocimientos sobre las opciones de compilación y enlace de librerías.

A lo largo de estos meses la gestión de la memoria usada durante la ejecución del programa ha jugado una carta importante. Las operaciones de reserva, liberación y acceso de memoria suponen recursos y tiempo para sistema, hacer un uso eficiente y escoger el momento adecuado para ellas es altamente importante. Para lograr hacer un uso más efectivo, hemos tenido que trabajar otro de los objetivos del proyecto, “Profundizar en la arquitectura ECS” el cuál nos ha permitido desarrollar un pequeño motor que gestione de forma correcta la información de nuestras entidades.

Uno de los puntos claves era estudiar y aprender nuevas técnicas de IA, en concreto el *Flocking* y el uso de los *Steering Behaviors*, además de poder crear una demo que nos permite mostrar el conocimiento adquirido.

Gracias a la lectura a conciencia sobre el tema y la dedicación de un largo periodo de tiempo para desarrollar algoritmos y funciones que trabaja estas técnicas, hemos conseguido entender su funcionamiento a la perfección y obtener resultados que satisfacen este altamente objetivo.

Al desarrollar todo únicamente en *C++17*, hemos conseguido apliar el conocimiento sobre el lenguaje y entender mejor el funcionamiento de algunas funcionalidades y herramientas incluidas en la *STL* y a desarrollar las nuestras propias, como puede ser el ya mencionado tipo de dato en coma fija, siguiendo una serie de normas y reglas estandarizadas.

6.1. Lecciones aprendidas

Después de estos meses trabajando en el Trabajo Final de Grado (TFG), hay una serie de mensajes y/o ideas que han ganado peso y que es puede ser interesante tener en cuenta.

A la hora de desarrollar un proyecto, herramientas y códigos hay miles, pero lo que siempre vas a tener es una máquina que lo interprete y ejecute. Conocer la máquina y como gestiona tu trabajo te permitirá ser más óptimo y encontrar fallos más rápido de haberlos, por ello es importante invertir tiempo en entender como funciona a bajo nivel.

Un pilar fundamental de la ingeniería son las matemáticas, es importante no descuidarlas y entender como hacer un uso correcto de ellas aplicado a la informática. Una vez más, entendiendo como las gestiona y representa internamente.

Es importante planificar el desarrollo con períodos de tiempo realistas, las prisas y la mala planificación llevan a soluciones temporales y malos funcionamientos.

Cuanto más general y flexible sea el código, mejor. Diseñar las herramientas de forma que sean reutilizable te facilitará el trabajo futuro y te hará plantear de mejor forma su funcionamiento.

Internet cuenta con mucha información y gente dispuesta a ayudar, pero no hay que olvidarse de los libros como fuente de un conocimiento más completo y profundo.

6.2. Trabajos futuros

Con el fin de evaluar que aspectos del proyecto quedan por finalizar, vamos a elaborar dos listas con elementos técnicos y de diseño que serían deseables para en el proyecto. La primera serán elementos que estimamos de alta prioridad y necesarios, que por motivos de tiempo y desarrollo no se han posido integrar a tiempo, la segunda serán cuestiones que es altamente probable que no se puedan realizar, que tienen una baja prioridad por el momento o que estan fuera de los objetivos generales del trabajo y por eso se han quedado fuera del desarrollo realizado hasta la fecha.

Objetivos principales:

- Dar *feedback* al jugador del estado de las unidades en el nivel.
- Crear un sistema de menús para la ejecución del juego.

Objetivos deseables:

- Ampliar el uso de OpenGL e intergrar sistema de carga de imagenes para poder añadir elementos visuales a las entidades, escenario y menús.

- Obtener/diseñar recursos gráficos apropiados para el proyecto.
- Integrar un sistema de sonido para poder reproducir al menos una melodía.

Bibliografía

- Amordron (2019). TaB AI discussion. <https://steamcommunity.com/app/644930/discussions/0/1649917420753004999/>.
- Andygmb (2014). Age of empires II AI Scripting. <https://gist.github.com/Andygmb/1e3a6d9d444b2dfa8c40>.
- Archive.org (2021). Página de Archive.org. https://archive.org/details/tfg_borja_ia.
- Bresenham (1962). Algoritmo de Bresenham. https://es.wikipedia.org/wiki/Algoritmo_de_Bresenham.
- Francis, B. (2020). Rebuilding a classic in Age of Empires II: Definitive Edition. https://www.gamasutra.com/view/news/358215/Rebuilding_a_classic_in_Age_of_Empires_II_Definitive_Edition.php.
- Francisco J. Gallego-Durán and Glenn Fiedler and Fred Howell and Alessandro Gatti (2019). TinyPTC. <http://bit.ly/tinyPTC-UA19>.
- Github (2021). Repositorio de GitHub del TFG. https://github.com/Boruha/TFG-IA_Grupal.
- Millington, I. (2009). *Artificial intelligence for games*. Morgan Kaufmann/Elsevier, Burlington, MA.
- Numantian (2019). They are Billions. https://store.steampowered.com/app/644930/They_Are_Billions/.
- Omar 'ocornut' (2021). Dear ImGUI. <https://github.com/ocornut/imgui>.
- Pikachunet (2018). TaB Informal level editor. https://www.reddit.com/r/TheyAreBillions/comments/9zvffg/example_of_number_of_zombies_per_wave_800/.
- Pritchard, M. (2000). Postmortem: Ensemble Studio's Age of Empires II: Age of Kings. https://www.gamasutra.com/view/feature/131844/postmortem_ensemble_studios_age_.php?page=1.
- Redmechanic (2017). A guide to scripting your own AI for AoEII. <https://steamcommunity.com/sharedfiles/filedetails/?id=1238296169>.

- Reynolds, C. (1986). Boids: Background and update. <http://www.red3d.com/cwr/boids/>.
- Sucasas, A. (2018). Entre las bambalinas de 'They Are Billions'. <https://www.xataka.com/videojuegos/entre-bambalinas-de-they-are-billions-el-nuevo-bombazo-del-videojuego-espanol>.
- Wikipedia (2020). Age of Empire II. https://en.wikipedia.org/wiki/Age_of_Empires_II.