UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Videojuego PAWN. Implementación de la inteligencia artificial y las animaciones

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Ramos Nebot, Diego

Tutor/a: Mollá Vayá, Ramón Pascual

CURSO ACADÉMICO: 2021/2022

Resumen

Pawn es un videojuego de acción y plataformas cuyo desarrollo empezó en la asignatura “Desarrollo de videojuegos 3D”. En este videojuego somos Pawn, un peón negro que se rebela contra su propio bando y se dispone a derrocar a su tirano rey.

En este TFG se plantea la creación de la inteligencia artificial de los enemigos de dicho juego, utilizando para ello la ampliación del Asset GAIA que incorpora árboles de comportamiento mediante la API PandaBT. También se implementarán las animaciones de dichos enemigos de tal forma que sean coherentes con sus estados.

Los enemigos incluidos en el juego son aquellos que ya forman parte del primer entregable, además de los añadidos en este trabajo. Siguiendo la línea argumental del juego, estos serían piezas de ajedrez, con patrones de ataques distintos a los enemigos anteriores, pero manteniendo componentes en común como podrían ser la patrulla o la persecución.

Por otra parte, también se incluirá en el TFG aquellos apartados artísticos en los que se ha trabajado de manera cooperativa con los demás desarrolladores (Jennifer Canuto Soler y Sergio Muñoz Alejandre), como puede ser el diseño y el modelado 3D o la creación de efectos visuales, aunque debido a la falta de experiencia este apartado tendrá un menor peso en el trabajo.

Palabras clave: videojuego, Unity, inteligencia artificial, Panda BT, animación

Abstract

Pawn is an action-platform game whose development began in the “3D-Games development” subject. In this game we are Pawn, a black pawn that revolts against his own side and is willing to topple its tyrant king.

This TFG proposes the creation of the artificial intelligence in the game, using for it the GAIA Asset extension that includes behavior trees thanks to the Panda BT tool. It will also contain the enemies’ animations implementation, so that they are coherent with their states.

The enemies included in the game are those that were in the game after the first deliverable, plus the ones added in this work. Following the story line of the game, these would be chess pieces, with different attack patterns to the previous ones, but keeping common components such as the patrol or the chasing.

On the other hand, this TFG will also contain the artistic aspects in which the rest of the developers (Jennifer Canuto Soler and Sergio Muñoz Alejandre) and I have been working cooperatively, like 3D modelling and design or VFX. However, this will have less impact due to the lack of experience.

Keywords: videogame, Unity, artificial intelligence, Panda BT, animation

Índice de contenido

Índice de figuras

1 Introducción

La inteligencia artificial se suele definir como aquella parte de la informática que intenta imitar el comportamiento o la inteligencia humana. Este ha sido uno de los campos más populares en los últimos años, debido en gran parte al *machine learning* y al aumento de aplicaciones de este conjunto de técnicas que, mediante el aprendizaje y el entrenamiento, consiguen mejorar enormemente los resultados.

Por otro lado, la definición de inteligencia artificial en los videojuegos varía, englobando el comportamiento de todas aquellas entidades del juego que buscan un objetivo, desde enemigos hasta la generación procedural de terrenos.

En este TFG se hablará principalmente del comportamiento de los distintos enemigos que componen Pawn, los cuales fueron programados y posteriormente adaptados para que funcionasen aprovechando las utilidades que ofrece la API (*Application Programming Interface*) de GAIA (*Game Artificial Intelligence Asset*) ampliada con *Behaviour Trees* (BT).

1.1 Motivación

La industria de los videojuegos en España lleva siendo un sector en alza durante muchos años, siendo la primera opción de ocio audiovisual del país. Incluso durante la reciente época de pandemia hubo un impulso significativo en el consumo de videojuegos, especialmente en la compra digital, debido entre otros factores al confinamiento y a la restricción de otros medios de entretenimiento sociales.

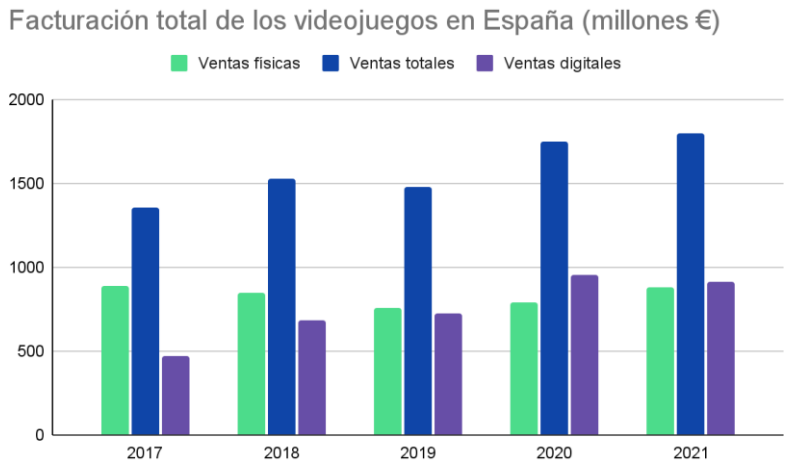

Figura 1 Ventas de videojuegos en España en los últimos años. Fuente propia.

La Figura 1 muestra una comparativa de las ventas de videojuegos en formato físico y digital en España, creada a partir de datos ofrecidos por la Asociación Española de Videojuegos (AEVI)[1].

Pese a la importancia e influencia de este sector en la cultura española, no ha sido hasta este año que la Comisión de Cultura del Congreso de los Diputados ha aprobado una reforma de ley que incluye la regulación de la conservación de videojuegos en la Biblioteca Nacional, convirtiéndolos así en patrimonio cultural nacional por ley[2]. De esta forma, si se aprueba por el Senado, se almacenará una copia digital (y física, si la hay) de todo videojuego desarrollado en el país. Y es que en los últimos años se ha visto a videojuegos españoles obtener reconocimiento no solo a nivel nacional, sino también mundial. Sin ir muy lejos, encontramos los galardonados *Gris*, de Nomada Studio, que obtuvo el premio de "Juego de mayor impacto" en los Game Awards de 2019[3], y *Metroid Dread*, desarrollado por el estudio MercurySteam junto con Nintendo, nominado a "Juego del año" y ganador de la categoría de "Mejor juego de acción/aventura" en los Game Awards de 2021[4].

Sin embargo, pese al auge de la industria de los videojuegos en España, existe una gran competitividad en la búsqueda de trabajo en este sector, por lo que es recomendable (y casi indispensable) tener experiencia anterior y contar con algún proyecto en tu expediente que respalde tus conocimientos.

Así pues, la razón por la que he decidido hacer este TFG es mi intención de dedicarme profesionalmente al desarrollo de videojuegos. Desde que era pequeño estos han sido mi pasión, y en los últimos años he descubierto que también disfruto haciéndolos.

1.2 Objetivos

Cuando se redactó el *Game Design Document* (GDD) del juego[5] para la asignatura de Desarrollo de Videojuegos 3D (D3D) se contempló la idea de dividirlo en seis niveles: el primero se centraría en los peones; el segundo en los caballos; el tercero en los alfiles; el cuarto en las torres; el quinto constaría de todas las piezas anteriores; y en el sexto encontraríamos el enfrentamiento final con el rey y la reina. Durante esta asignatura se llegó a desarrollar el nivel uno, el cual consistía en escapar de una jaula

[1] El Anuario del Videojuego – Documentación http://www.aevi.org.es/documentacion/el-anuario-del-videojuego/
[2] Proyecto de Ley por la que se modifica la Ley 23/2011, de 29 de julio, de depósito legal https://www.congreso.es/actualidad/sesiones-de-comisiones?idOrgano=377&idSesion=20&fecha=22%2F03%2F2022
[3] The Game Awards 2019 https://youtu.be/jxAihuiYxuU?t=7481
[4] The Game Awards 2021 https://youtu.be/OS4m2O3V93o?t=8267
[5] Disponible como anexo

y vencer a todos los enemigos que se encontraban en el mapa para después derrotar al jefe final del nivel, un peón más grande y fuerte que el resto.

Los objetivos principales para la segunda parte y que engloba este TFG son:

- Diseño y creación del nivel 2, de manera que sea la continuación al trabajo previo realizado en el primer cuatrimestre. Este estaría dividido en tres partes, siendo la zona del jefe final la última y la que se detalla en esta memoria.
- Creación de los modelos 3D y animaciones necesarias para el alcance previsto del proyecto.
- Creación de la inteligencia artificial necesaria para el alcance previsto del proyecto, utilizando para ello la API GAIA con PandaBT.

1.3 Colaboraciones

Este trabajo describe parte del desarrollo del videojuego Pawn, un proyecto que nace en la asignatura D3D. Esta asignatura proponía una docencia interdisciplinar en la que se creaban equipos formados tanto por alumnos del Grado en Ingeniería Informática de la ETSIINF como del Grado en Diseño y Tecnologías Creativas de BBAA. El propósito de esta colaboración no era solo obtener un resultado más compacto, sino también fomentar la comunicación entre las distintas áreas de trabajo con tal de simular un entorno más similar a los equipos de desarrollo de videojuegos que encontramos en la realidad.

El desarrollo de la primera parte del proyecto se realizó junto con Pablo Moreno Martínez, Siboney Luis Martín, Luis Alberto Álvarez Zavaleta, David Arnal García, Jennifer Canuto Soler y Sergio Muñoz Alejandre, siendo estos dos últimos con quién se realiza la segunda parte. Los títulos de sus TFG son “Videojuego Pawn: Multiplataforma, desarrollo de mecánicas y efectos de sonido” y “PAWN: Del tablero a videojuego en 3D con Unity. Desarrollo del modo multijugador online”, respectivamente.

A continuación, se muestra una tabla con la implicación de todos los componentes del grupo a lo largo del desarrollo de Pawn.

Miembros	**Facultad**	**Participación en la primera parte**	**Participación en la segunda parte**
Pablo Moreno Martínez	BBAA	Modelado 3D y animación	-

Siboney Luis Martín	BBAA	Diseño 2D y personajes	-
Luis Alberto Álvarez Zavaleta	ETSIINF	Gestión de interfaces, configuración del juego	-
David Arnal García	ETSIINF	Gestión de interfaces, configuración del juego	-
Jennifer Canuto Soler	ETSIINF	Diseño de niveles, música y sonido	Portabilidad del juego, realidad virtual, jugabilidad, sonido y diseño de niveles
Sergio Muñoz Alejandre	ETSIINF	Programación y diseño de niveles	Modo multijugador online, diseño de niveles, página web[6]
Diego Ramos Nebot	ETSIINF	Inteligencia artificial y programación	Inteligencia artificial, modelado 3D, animación, diseño de niveles

[6] Pawn https://pawngame.es/#

2 Tecnología utilizada

Durante el transcurso de este proyecto se han usado distintas herramientas que han permitido desarrollar el juego, modelar los personajes, crear animaciones y redactar la memoria:

- **Unity**: Motor de videojuegos sobre el que se ha realizado el proyecto.
- **Blender**: Software que permite hacer modelado 3D, *rigging*[7], animación y renderizado. En este proyecto se utilizó para modelar y animar algunos enemigos, crear y editar modelos 3D y grabar animaciones para el jugador.
- **Google Docs**: Editor de texto en línea que se usó para escribir la memoria y mantener una copia de ésta en la nube por precaución.
- **Microsoft Word**: Editor de texto que se utilizó para la versión final de la memoria y exportarla en formato *pdf*.
- **Lucidchart**: Página web para crear diagramas. Usado para algunas de las imágenes de esta memoria.
- **Visual Studio 2019**: Entorno de desarrollo usado principalmente para realizar toda la programación implementada en *Unity*.
- **GitHub**: Plataforma de control de versiones y colaboración. Se utilizó para subir los cambios realizados de manera local para que el resto de los compañeros pudiesen actualizar sus proyectos y viceversa.
- **Asset Store**: Plataforma de *Unity* que se utiliza para descargar *assets* e integrarlos en el proyecto. Desde esta tienda se han descargado modelos 3D, APIs, efectos especiales, etc.
- **NavMesh**: API que permite convertir *gameObjects*[8] en agentes capaces de moverse a través de mallas. Es la base del comportamiento de los enemigos de este juego.
- **GAIA**: API desarrollada por José Alapont y ampliada por Jorge Andreu Royo. Permite la creación de máquinas de estado y árboles de comportamiento. Es utilizada para el diseño de los enemigos.
- **Panda BT Free**: *Framework* de creación de árboles de comportamiento. En este proyecto está integrado dentro de GAIA y se usa para definir el comportamiento de los enemigos.
- **Discord**: Aplicación de mensajería instantánea. Utilizada para las reuniones con los miembros del equipo.

[7] Rigging https://docs.blender.org/manual/es/2.79/rigging/introduction.html
[8] GameObject https://docs.unity3d.com/es/530/Manual/class-GameObject.html

3 Estado del arte

3.1 Géneros de videojuegos

Antes de empezar el desarrollo de un videojuego, normalmente, se decide a qué género (o géneros) pertenecerá. Dicho género servirá como etiqueta para clasificarlo dentro del mercado con tal de atraer a los posibles compradores en la etapa de marketing.

Esta decisión tendrá un papel importante en los beneficios del propio juego, debido a que existen géneros predominantes entre los jugadores. La Figura 2 Gráfico que muestra géneros de videojuegos y su número de juegos y media de las ventas en el plazo 2019-2022muestra una lista con diferentes géneros y la cantidad de juegos de estudios independientes que se lanzaron pertenecientes a cada grupo en el plazo de enero de 2019 a abril de 2022 en Steam. La línea roja representa la media de los beneficios obtenidos en cada género. A partir de estos datos se puede deducir que la prioridad de los estudios más pequeños no suele ser la generación de ingresos, sino más bien el desarrollo de proyectos personales.

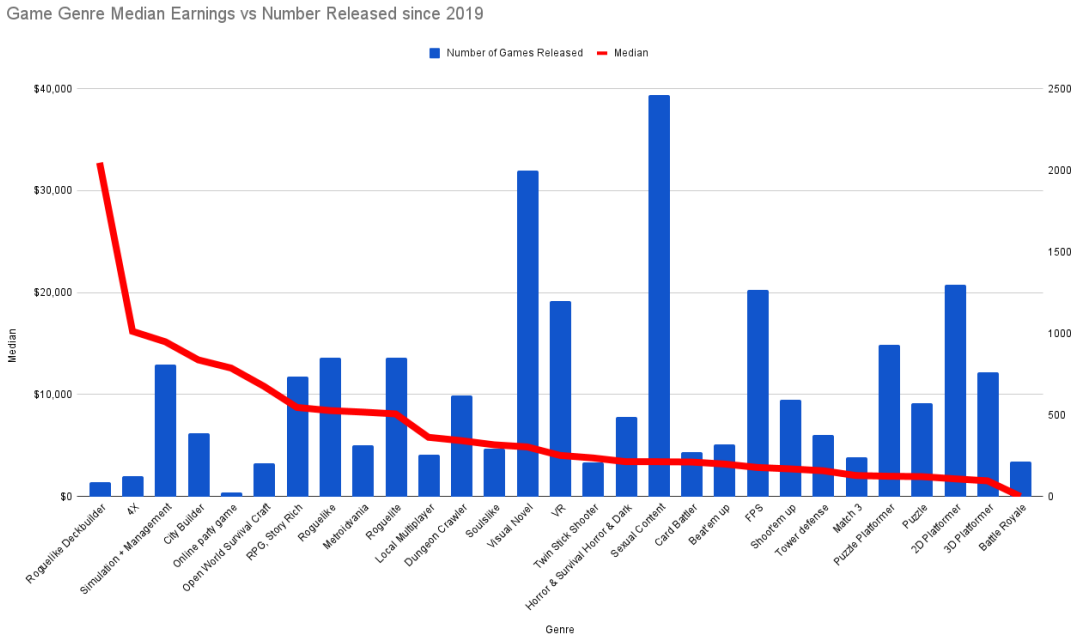

Figura 2 Gráfico que muestra géneros de videojuegos y su número de juegos y media de las ventas en el plazo 2019-2022[9]

[9] What genres are popular on steam in 2022 https://howtomarketagame.com/2022/04/18/what-genres-are-popular-on-steam-in-2022/

En el caso de Pawn, cuando se ideó la temática general se pensó como un juego de plataformas 3D, tomando como referencias otros como *Ratchet & Clank*, *Spyro the Dragon* o *Crash Bandicoot*. Sin embargo, conforme fue avanzando el desarrollo, terminó por convertirse en un juego más cercano al género de acción-aventura con elementos de plataformas 3D, pero en menor medida.

Figura 3 Ratchet & Clank: Armados hasta los dientes (2007)[10]

Si se utiliza la Figura 2 como referencia, Pawn podría incluirse dentro de “*Story rich*”, “*Online party game*” y “*3D platformer*”, debido a su narrativa, al modo online desarrollado por Sergio Muñoz y a los elementos de plataformas. Estos géneros albergan la séptima, quinta y vigesimoctava posición, respectivamente. Si se hiciera la media de estas posiciones, Pawn se encontraría aproximadamente en decimotercer lugar, con alrededor de 5.000$ de ingresos. Aunque el resultado final depende de más factores, esta estimación rápida sirve como un punto para tener en cuenta cuando se empieza a crear el concepto de un videojuego y qué se espera conseguir de éste.

Pawn, sin embargo, nació como un proyecto universitario y no como un producto que saldría al mercado, por lo que se eligió el género que más se acomodaba a los gustos de los desarrolladores y diseñadores.

3.2 Tecnologías disponibles

3.2.1 Motores de videojuegos

Actualmente existen varias opciones a la hora de crear un videojuego, y una de las primeras decisiones es en qué motor desarrollarlo. Aunque es posible crear uno desde

[10] Ratchet Galaxy https://ratchet-galaxy.com/en/games/ps3/ratchet-and-clank-future-tools-of-destruction/media/screenshots

cero, esta elección se suele reservar a estudios con un presupuesto elevado que pueden permitirse invertir dinero y tiempo en hacerlo, normalmente porque buscan tener un sistema de físicas o unas especificaciones muy concretas.

Algunos de estos ejemplos son el *Hedgehog Engine*, creado por el Sonic Team y usado principalmente para juegos de Sonic, o el *Creation Engine* de Bethesda, motor de juegos como *The Elder Scrolls V: Skyrim*. Estos motores no son de un solo uso, sino que se conservan y evolucionan para adaptarlos a nuevos desarrollos, creando nuevas versiones si es preciso.

Figura 4 Sonic Unleashed (2008) en el motor Hedgehog Engine.[11]

Sin embargo, en los últimos años se ha visto la aparición y popularización de motores gratuitos que permiten crear videojuegos y obtener resultados muy buenos. Esto ha provocado un aumento notable en el lanzamiento de títulos por parte de estudios independientes o incluso individuales. Esta sección estudia algunas de estas opciones y su comparación.

- **Unity**: Es un motor de juegos multiplataforma, siendo compatible con Windows, Mac, Linux, iOS, Android, Nintendo Switch, Xbox y PlayStation, entre otras plataformas[12]. En los últimos años se ha convertido en una de las opciones más elegidas tanto por empresas grandes como por estudios independientes, gracias a su amplia funcionalidad y capacidad de crear videojuegos de todos los géneros. Permite crear juegos tanto en 2D como en 3D y utiliza el lenguaje C#.

[11] Sonic Unleashed – Chun Nan Day https://www.youtube.com/watch?v=8tTDg4LLRCY&ab_channel=Waffle.
[12] What platforms are supported by Unity? https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity-

Entre sus utilidades destaca el Asset Store, tienda online que sirve para descargar desde librerías hasta *assets* 3D, proporcionados tanto por Unity Technologies como por miembros de la comunidad.
Aunque es gratuito, también ofrece otros planes de pago que incluyen más funcionalidades, empezando por 399$ al año. Si se utiliza la versión gratuita y los beneficios superan los 100.000$ será necesario obtener un plan de pago[13].

Figura 5 Syberia: The World Before (2022) en Unity.[14]

- **Unreal Engine 4**: Otra de las opciones más populares en la actualidad es este motor propiedad de Epic Games. Al igual que *Unity*, es compatible con la gran mayoría de las plataformas de videojuegos más utilizadas: Mac, Linux, Windows, PlayStation, Xbox, Nintendo Switch, Android…[15] También permite crear juegos en 3D y 2D, pero utiliza el lenguaje C++. Además, dispone de un sistema de programación visual llamado Blueprints, que proporciona una interfaz basada en nodos con la que se pueden crear elementos de *gameplay*. Permite el acceso al código fuente, pero no es *open-source*. Esto quiere decir que se puede modificar la base del motor en un proyecto propio, pero no se puede publicar el código fuente.
  Unreal Engine es gratuito, pero hay que pagar un 5% en regalías a partir del millón de dólares generado, sin incluir los beneficios hasta ese momento. Por ejemplo, si un juego obtiene un beneficio de 2 millones de dólares, pagaría 50.000 dólares, correspondiente al 5% del millón extra obtenido por encima del umbral[16].

[13] Financial Thresholds https://unity3d.com/es/legal/terms-of-service/software#:~:text=The%20Financial%20-Threshold%20for%20Unity,for%20internal%20projects%20or%20prototyping
[14] gamingcoffee https://www.gamingcoffee.com/2022/02/se-revelan-fotos-del-gameplay-de-syberia-the-world-before/
[15] Multi-platform development https://www.unrealengine.com/en-US/features/multi-platform-development
[16] How much do I have top pay for Unreal Engine? https://www.unrealengine.com/en-US/faq

La curva de aprendizaje en este motor tiende a ser mayor que en *Unity*, ya que no es tan intuitivo de utilizar. No obstante, su principal diferencia y ventaja respecto a otros motores es su capacidad gráfica. *Unreal* soporta un renderizado más rápido, y el recientemente lanzado *Unreal Engine* 5 cuenta con grandes avances tecnológicos en cuanto a iluminación y carga poligonal se refiere[17].

Figura 6 Returnal (2021) en Unreal Engine.[18]

- **Godot**: Se trata de un motor gratuito y open-source (bajo licencia del MIT[19]), y es famoso sobre todo entre principiantes. Es considerado uno de los mejores motores en el mercado para la creación de juegos 2D, aunque también permite hacer 3D.

  A diferencia de los anteriores, los juegos desarrollados en este motor no están ligados a suscripciones de pago o regalías. Utiliza el lenguaje GDScript, cuya sintaxis es similar a Python.

  Las plataformas compatibles de este motor son Linux, Mac, Windows, Android, iOS y otras plataformas web y de realidad virtual[20]. Sin embargo, el soporte a videoconsolas tiene que ser a través de terceros debido a su naturaleza *open-source*, ya que como indica su página web[21], para poder desarrollar en consolas es necesario tener una licencia como empresa y utilizar SDKs (*software development kit)* específicos, los cuales están protegidos por acuerdos de confidencialidad.

[17] Unreal Engine 5.0 Release Notes https://docs.unrealengine.com/5.0/en-US/unreal-engine-5-0-release-notes/
[18] Returnal Screenshots (10) https://www.pushsquare.com/games/ps5/returnal/screenshots#enlarge-7
[19] License https://godotengine.org/license
[20] Features https://godotengine.org/features
[21] Console support in Godot https://docs.godotengine.org/en/latest/tutorials/platform/consoles.html

Uno de los puntos fuertes de este motor respecto a otros es su propia comunidad de desarrolladores, pues es muy activa y consta de gran cantidad de foros donde se resuelven dudas y problemas en general.

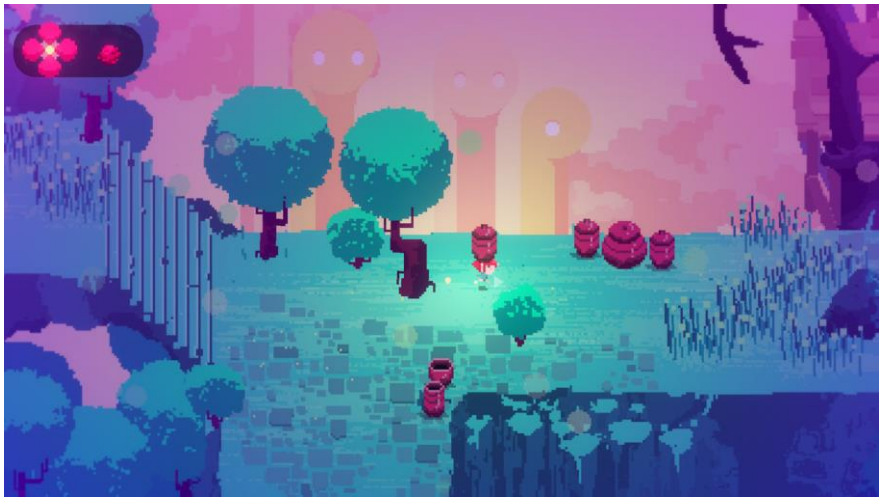

Figura 7 Lila's Sky Ark (2022) en Godot.[22]

- **CryEngine**: Perteneciente a la empresa alemana Crytek, *CryEngine* es otro de los motores gráficos más populares de la época actual. Aunque hace varios años había que pagar una licencia, a partir de 2016 pasó a ser gratuito y compartió su código fuente[23].

  En cuanto a su modelo de negocio, es similar a *Unreal Engine*, ya que hay que pagar un 5% en regalías, en este caso de los beneficios anuales y sin contar los primeros 5.000$[24]. Por ejemplo, si un juego obtiene 100.000$ en un año, tendría que pagar 4.750$. Esta no es su única similitud con *Unreal*, y es que también destaca por su capacidad de crear entornos visualmente realistas.

  Su lenguaje de programación es Lua, y permite la opción de llamar a código en C++. Las plataformas soportadas por el motor son Windows, Linux, PlayStation 4, Xbox One, Oculus Rift, OSVR, PSVR, y HTC Vive[25], lo que le otorga un buen puesto en plataformas donde desarrollar videojuegos de realidad virtual.

  Tal es la fama de este motor que su primera versión fue adquirida por Ubisoft, que la amplió y modificó para desarrollar juegos de la saga *Far Cry* bajo el nombre de *Dunia Engine*. Además, en 2015 cedió una licencia a Amazon, y un año después salió a la luz el motor *Amazon Lumberyard*, otra derivación de

[22] Programas descargables Nintendo Switch - Lila’s Sky Ark https://www.nintendo.es/Juegos/Programas-descargables-Nintendo-Switch/Lila-s-Sky-Ark-2192606.html#Galer_a

[23] Building the Engine from Source Code https://docs.cryengine.com/display/CEPROG/Building+the+Engine+from+-Source+Code#:~:text=The%20CRYENGINE%20source%20code%20is,browser%20by%20clicking%20this%20link

[24] Licensing https://www.cryengine.com/support/view/licensing

[25] Platform support https://www.cryengine.com/support/view/general#:~:text=on%20our%20website.-,Platform%20support,Mobile%20support%20is%20in%20development

CryEngine que se convirtió en una de las muchas opciones para desarrollar videojuegos gracias a su acceso gratuito y su naturaleza multiplataforma. Sin embargo, al contrario que *CryEngine*, está limitado a juegos 3D.
En 2021 Amazon se asoció con la Linux Foundation y sacó una versión de Lumberyard bajo el nombre de *Open 3D Engine* (O3DE), la cual es *open-source*[26].

Figura 8 Sniper Ghost Warrior Contracts 2 (2021) en CryEngine.[27]

- **GameMaker**: Creado por Mark Overmars en 1999 y desarrollado por YoYo Games desde 2007, este motor está ideado para aquellos que quieran crear juegos sin tener necesariamente un amplio conocimiento de programación. Proporciona un lenguaje de programación visual y otro para scripts, similar a C. Esto ayuda a que el proceso de crear juegos en esta plataforma sea muy rápido comparado a otros, pero también limita sus capacidades. Está orientado al desarrollo de juegos en 2D, aunque también se pueden crear en 3D.
  Se puede obtener y utilizar de forma gratuita. No obstante, se requieren distintos niveles de suscripción para poder publicar los juegos: para plataformas de escritorio (Windows, macOS y Linux) es de 4.25€; para el lanzamiento en web (HTML5) y dispositivos móviles (iOS, Android, Amazon Fire, Android TV y tvOS) es de 8.19€; y para consolas (PlayStation 4 y 5, Xbox One, Xbox Series X|S y Nintendo Switch) es de 67.99€[28]. Los niveles superiores incluyen las ventajas de los inferiores.

[26] Amazon Lumberyard https://aws.amazon.com/es/lumberyard/
[27] Steam Store – Sniper Ghost Warrior Contracts https://store.steampowered.com/app/1338770/Sniper_Ghost_Warrior-_Contracts_2/?l=spanish
[28] Suscripciones a GameMaker Productos https://gamemaker.io/es/get

Figura 9 Undertale (2015) en GameMaker.[29]

A continuación, se muestra una tabla comparativa de los anteriores motores, con los puntos que se tuvieron en cuenta para la realización de Pawn.

	Unity	**Unreal Engine**	**Godot**	**CryEngine**	**GameMaker**
Multiplataforma (Windows y Android)	Sí	Sí	Sí	Windows sí Android no	Sí
Multijugador online	Sí	Sí	Sí	Sí	Sí
Realidad virtual (Google Cardboard)	Sí	Sí	No	No	No
Precio	Gratuito, con opciones de pago	Gratuito	Gratuito	Gratuito	Gratuito, con opciones de pago
Facilidad de uso	Fácil	Media	Fácil	Media	Muy fácil
3D/2D	Ambos	Ambos	Ambos	Ambos	Ambos

Como se puede ver en la tabla, las opciones más válidas para el desarrollo de Pawn eran *Unity* y *Unreal Engine*. Aunque la decisión ya estaba tomada desde que en la asignatura de D3D se empezó a usar *Unity*, este motor seguiría siendo la mejor opción

[29] Programas descargables Nintendo Switch https://www.nintendo.es/Juegos/Programas-descargables-Nintendo-Switch/UNDERTALE-1347694.html#Galer_a

debido a que tiene una curva de aprendizaje menos pronunciada y posee una cantidad superior de *assets* en comparación a *Unreal Engine*.

3.2.2 Software de modelado 3D

Por otro lado, las principales herramientas de modelado 3D con las que se puede usar *Unity* y están disponibles en Windows son Blender, 3ds Max, Maya, Modo, ZBrush, Cinema 4D y Adobe Substance. Para este caso, el estudio de las distintas alternativas se simplifica mucho, y se resume en que Blender es el único software gratuito capaz de realizar todas las funciones que se necesitaban para este proyecto[30].

	Blender	**3ds Max**	**Maya**	**Modo**	**ZBrush**	**Cinema 4D**	**Adobe Substance**
Animación y rigging	Sí	Sí	Sí	Sí	No	Sí	No
Gratuito	Sí	No	No	No	No	No	No

3.2.3 Máquinas de estado y árboles de comportamiento

Por último, queda pendiente estudiar las dos posibilidades que ofrece la API de GAIA: máquinas de estado finitas (FSM) y árboles de comportamiento. Para compararlos se utilizará como ejemplo práctico el comportamiento de los fantasmas del videojuego *Pac-Man*[31].

En el campo de la inteligencia artificial, las FSM son modelos de comportamiento compuestos por estados, transiciones y sus respectivas acciones. La entidad con la FSM asignada se encontrará en un estado en todo momento, y para cambiar a otro necesitará cumplir las condiciones que invocan a la transición. Aunque son relativamente sencillas de adaptar y escalar, tienden a adquirir mayor complejidad e ineficiencia conforme crece el número de estados. Se suelen utilizar para casos más visuales, como por ejemplo los estados de animación de un personaje (como se verá en el apartado 4.2.2.1).

[30] Blender Features https://www.blender.org/features/
[31] Pac-Man https://es.wikipedia.org/wiki/Pac-Man

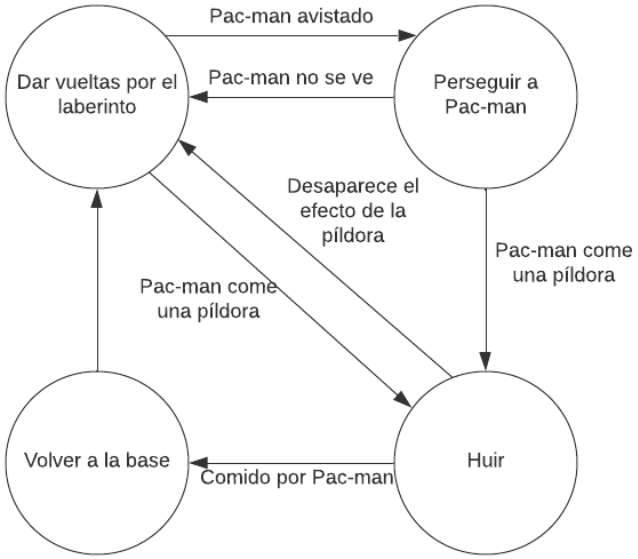

Figura 10 Máquina de estados del comportamiento de un fantasma del juego Pac-Man. Fuente propia.

Los BT, aunque similares a las FSM, se distinguen por su estructura modular dividida por nodos. Dichos nodos pueden devolver tres estados posibles: éxito, fracaso o *running*, si se está ejecutando. Su ventaja principal es su capacidad de diseñar modelos de comportamiento complejos a partir de tareas simples. Además, gracias a su flexibilidad, son muy sencillos de modificar y añadir cambios, y muchas de las herramientas incorporan editores visuales para ello. Una de las funcionalidades con la que cuentan y las FSM no tiene (sería necesario crear otra FSM) es ejecutar nodos de manera concurrente. Aunque GAIA cuenta con una versión de las FSM que permite la concurrencia, su dificultad respecto a los BT es mayor, puesto que en estos es solo necesario utilizar un nodo de tipo “paralelo”, como se verá en el apartado 5.1.2.

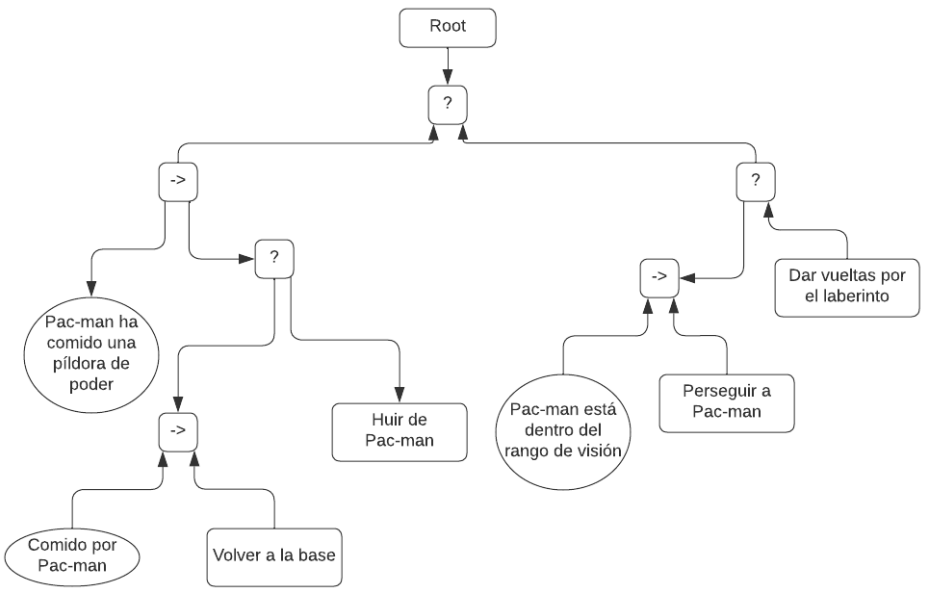

Figura 11 Behavior Tree de un fantasma del juego Pac-Man. Fuente propia.

En la Figura 11 los nodos con una flecha indican secuencia, es decir, ejecutará sus hijos de izquierda a derecha siempre que devuelvan éxito, por lo que, si uno fracasa, el padre devolverá fracaso. Los nodos con un interrogante implican selección, lo que quiere decir que devolverán éxito cuando el primero de sus hijos se ejecute con éxito.

Como se puede ver en la Figura 10, pese a tener solo cuatro estados, la máquina de estados cuenta ya con bastantes transiciones para tener en cuenta, mientras que el árbol de la Figura 11 está mejor organizado y a primera vista resultaría más fácil añadir una tarea nueva.

4 Desarrollo del proyecto

En este apartado se documentará toda la inteligencia artificial implementada en el juego, empezando por una introducción del *asset NavMesh*, encargado de dotar a los enemigos de movimiento y consciencia sobre el entorno. Posteriormente, se detallará el comportamiento de cada enemigo, así como si ha sido necesario la creación de algún tipo de modelo 3D o animación para dicho enemigo.

4.1 NavMesh

Se trata de una clase disponible en el módulo *Unity.AI*, descargable a través de GitHub[32]. Esta herramienta permite utilizar técnicas de búsqueda de caminos para inteligencias artificiales. Los componentes más importantes y de los cuales se hace uso en este proyecto son los siguientes:

- **NavMesh Surface**: Es el componente encargado de hacer el *bake*[33] del terreno para generar las superficies por las que se pueden mover los agentes. Se puede crear una por cada tipo de agente, y en el proceso se indican las capas de los objetos que podrán ser tratados como superficie transitable y si se hará a partir de sus mallas o de sus colisionadores.

Figura 12 Horneado de un terreno hecho con el componente NavMesh Surface. Fuente propia.

- **NavMesh Agent**: Es el componente que se añade a las entidades que se desea que tengan movimiento. Tiene parámetros como velocidad, velocidad angular, aceleración, altura, etc. Desde código se le puede asignar un destino

[32] NavMeshComponents https://github.com/Unity-Technologies/NavMeshComponents
[33] NavMesh Baking https://learn.unity.com/tutorial/navmesh-baking#

usando el método *agent.SetDestination()*, que admite un vector tridimensional como argumento: la posición a la que ir.

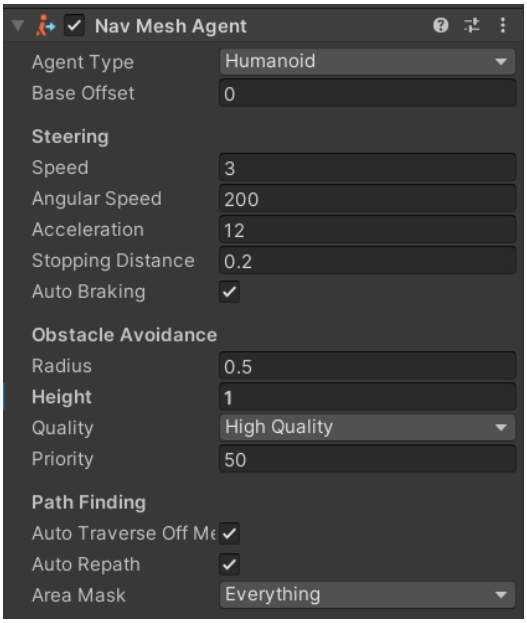

Figura 13 Ejemplo de NavMesh Agent. Fuente propia.

- **NavMesh Obstacle**: Si al hacer un horneado de la malla hay objetos que no tienen la capa asignada para poder ser procesados, estos serán ignorados por los agentes, que los atravesarán. Una posible solución es asignarles la capa correspondiente para poder ser horneados, pero esto podría provocar que los enemigos se subieran encima de objetos o intentaran acceder a lugares que no pueden alcanzar. Para poder solucionar este tipo de problemas existe este componente, el cual permite declarar una especie de colisionador que creará un agujero en la malla, por lo que los enemigos tratarán de bordearlo.

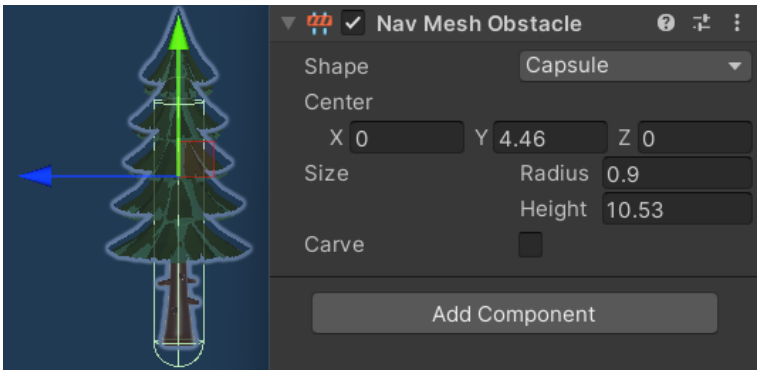

Figura 14 Ejemplo de NavMesh Obstacle en un árbol. Fuente propia.

Todos los enemigos del juego aprovechan esta clase para crear su movimiento y simular su comportamiento.

4.2 Enemigos

Los enemigos se desarrollaron en una escena[34] de pruebas para no interferir en el trabajo de los demás compañeros. Como se indicó en la introducción, existe uno por cada pieza de ajedrez, y a veces cuentan con más de una versión.

4.2.1 Peones

Los peones son los enemigos más básicos del juego y los primeros en ser implementados, debido a que son los protagonistas del nivel uno. Su comportamiento es el más sencillo de todos los enemigos, siendo muy similar al del jugador, ya que se trata del mismo tipo de pieza. Este enemigo consta de dos versiones: el peón básico y el peón jefe. Pese a ser desarrollados en la asignatura de D3D y no en este TFG, se utilizará al peón básico para explicar la base del resto de enemigos, ya que heredaron sus funciones principales.

4.2.1.1 Peón básico

Tiene una apariencia muy similar a la del protagonista, pero con otras expresiones faciales. Existen tres versiones diferentes, que, si bien tienen la misma conducta, varían sus parámetros para dotar al juego de más variedad. Estos se llaman peón flaco, peón alto y peón gordo, y aunque fueron diseñados en la primera parte, no fue hasta la segunda cuando se añadieron, ya que en el modo online no se distinguían del resto de jugadores, por lo que hubo que importarlos al juego desde Blender.

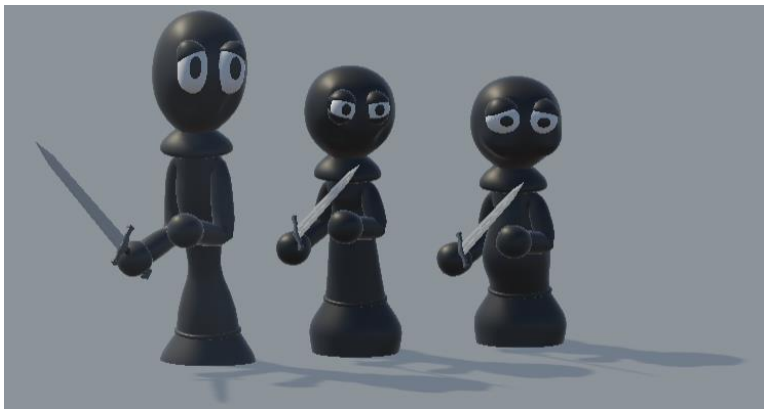

Figura 15 Peón flaco, peón alto y peón gordo respectivamente. Fuente propia.

La manera de atacar es igual a la del protagonista: una vez esté cerca del jugador dará un espadazo. En cuanto al comportamiento, el enemigo cambiará entre tres estados mientras esté vivo: patrullar, perseguir y atacar. Para cambiar de estado, el

[34] Scenes https://docs.unity3d.com/Manual/CreatingScenes.html

enemigo comprobará en el método *Update*[35] si hay un objeto etiquetado[36] como jugador dentro de diferentes rangos en forma de esfera a su alrededor.

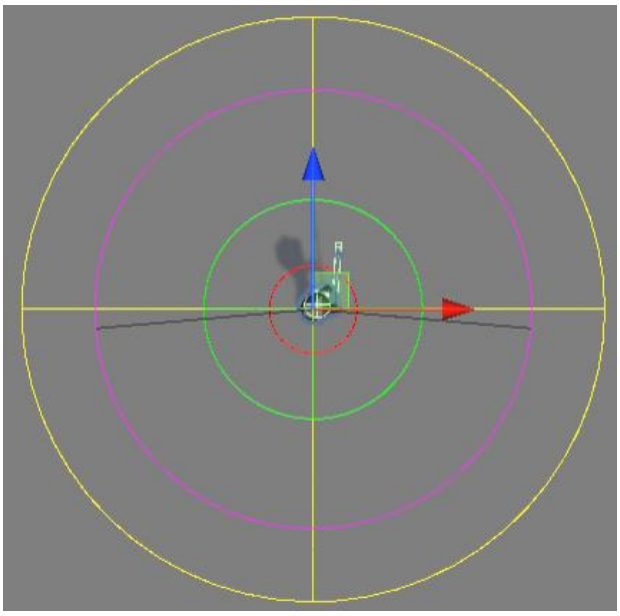

Figura 16 Vista superior de los rangos del peón. Fuente propia.

Como muestra la Figura 16, para el caso del peón encontramos los siguientes rangos:

- **Rango de campo de visión** (magenta y negro): Este consta de dos parámetros: un radio y un ángulo. Sirve para detectar al jugador una vez entre en el ángulo cerrado indicado. Para la implementación de esta mecánica se sirvió de la ayuda del tutorial de Comp-3 Interactive[37], el cual incluía cómo indicarle al enemigo qué capas de obstrucción son las que pueden cubrir al jugador y también cómo crear un editor que mostrase en la escena el ángulo de visión.
- **Rango de cercanía** (verde): Este rango es similar al anterior, pero sin contar con un ángulo. Al ser más pequeño que el campo de visión, su utilidad principal es indicarle al enemigo que el jugador está muy cerca de él, con tal de que no pueda derrotarlos si se acerca de espaldas.
- **Rango de alcance de visión** (amarillo): Una vez el enemigo vea al jugador, este rango servirá para perseguirlo siempre y cuando esté dentro de él, por lo que suele ser algo mayor que el de campo de visión.

[35] MonoBehaviour.Update() https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html
[36] Layers https://docs.unity3d.com/Manual/Layers.html
[37] How to Add a Field of View for Your Enemies [Unity Tutorial] https://www.youtube.com/watch?v=j1-OyLo77ss&ab_channel=Comp-3Interactive

- **Rango de ataque** (rojo): Como su propio nombre indica, es el rango que provocará que el enemigo ataque al jugador una vez entre en él.

Así pues, la máquina de estados del peón quedaría de la siguiente forma:

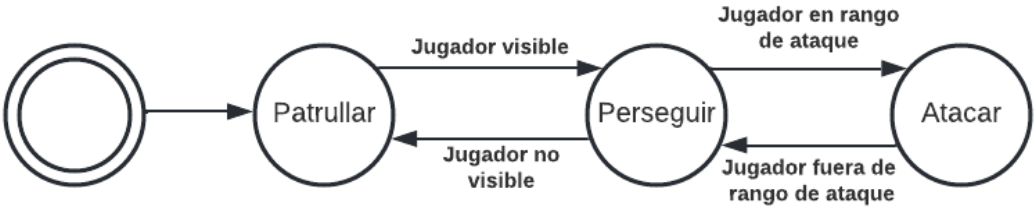

Figura 17 Máquina de estados del peón. Fuente propia.

En el estado de patrulla, el peón enemigo caminará por el campo de batalla, y se le puede indicar si lo hará de manera aleatoria o entre distintos puntos. Dependiendo de los argumentos que se le pase en el *Inspector*[38] se darán tres casos:

- **Ningún punto**: El enemigo patrullará entre unos puntos en el mapa que generará él mismo de manera aleatoria dentro de un rango específico.
- **Un punto**: El enemigo se quedará quieto en ese punto mirando en la dirección en la que se ha puesto en la escena.
- **Dos o más puntos**: El enemigo patrullará entre esos puntos, siendo posible que pueda pararse un tiempo específico en ellos si se quiere.

Cuando muera, el enemigo se detendrá y reproducirá la animación de muerte. Al mismo tiempo, se ejecutará una *corrutina*[39] que reducirá el canal alfa de los materiales de los enemigos en cada fotograma para dar la sensación de que desaparecen, y se invocará un método para destruir el objecto que contiene el script del enemigo.

Además de poder parametrizar los rangos y los puntos de patrulla, también hay otras variables personalizables como el tiempo entre ataques, la velocidad de persecución, el daño del ataque o la vida, y como se ha mencionado anteriormente, el componente *NavMesh Agent* también permite cambiar otros valores como la velocidad del agente, la aceleración o la velocidad angular, como mostraba la Figura 13.

El resto de las piezas tomarán como referencia las características base del peón, heredando principalmente el estado de patrulla y el cambio a otros estados según la

[38] Usando el Inspector https://docs.unity3d.com/es/530/Manual/UsingTheInspector.html
[39] Corrutinas https://docs.unity3d.com/es/530/Manual/Coroutines.html

distancia respecto al jugador. No obstante, tendrán una conducta más compleja e intentarán diferenciarse de los peones, creando patrones de ataques originales, no solo para distinguirse del movimiento del jugador, sino también para crear una curva de dificultad que sea coherente tanto con el apartado narrativo del juego como con la cantidad de niveles que se espera desarrollar en el futuro.

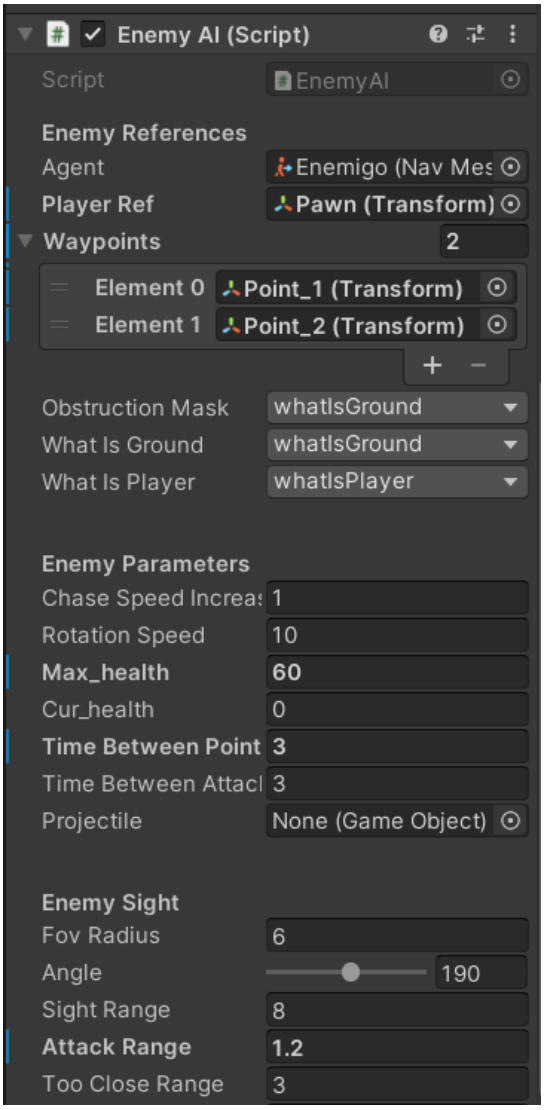

Figura 18 Parámetros del script del peón. Fuente propia.

4.2.2 Caballos

Los caballos aparecen a partir del nivel dos. A diferencia de los enemigos posteriores, para esta pieza también fue necesario crear el jefe, ya que formaba parte del modo campaña.

Puesto que no dio tiempo a ser diseñados en la primera parte, fue necesario crear los modelos 3D a partir de la ilustración de Siboney Luis Martín, además de grabar las

animaciones necesarias para los distintos movimientos de las piezas. Para ello se utilizó el programa Blender, ya que se contaba con experiencia anterior en la aplicación. Para los jinetes, sin embargo, se utilizó uno de los modelos descartados que creó Pablo Moreno Martínez con tal de ahorrar tiempo en el apartado artístico.

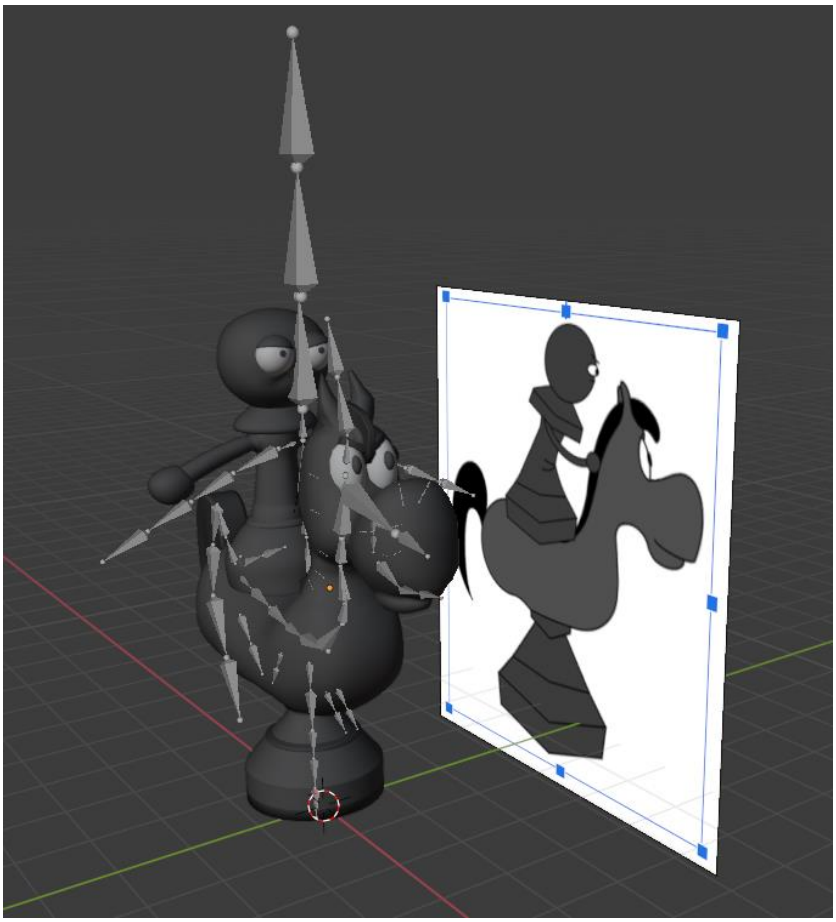

Figura 19 Modelo del caballo en Blender. Fuente propia.

En la Figura 19 se puede apreciar una técnica útil y que fue utilizada para convertir imágenes en 2D a 3D, que consiste en poner la imagen de referencia sobre uno de los planos XYZ para posteriormente empezar a moldear el personaje en vista ortográfica.

También se puede observar que los personajes tienen una especie de esqueleto; este puede crearse desde cero (como el del jinete) o utilizar uno predeterminado y modificarlo después (como el del caballo), y tiene dos funciones principales:

- Sirve para indicar a la malla que compone el modelo cómo deformarse cuando reproduce las animaciones. Para ello se puede cambiar al modo de *Weight Paint*[40], donde podemos seleccionar uno de los huesos del esqueleto de manera individual y pintar el cuerpo del modelo con valores entre 0 y 1 como si

[40] Weight Paint https://docs.blender.org/manual/en/latest/sculpt_paint/weight_paint/introduction.html

de un mapa de temperaturas se tratase, siendo el color más frío el que no modificaría la malla y el más cálido el que más la deformase.

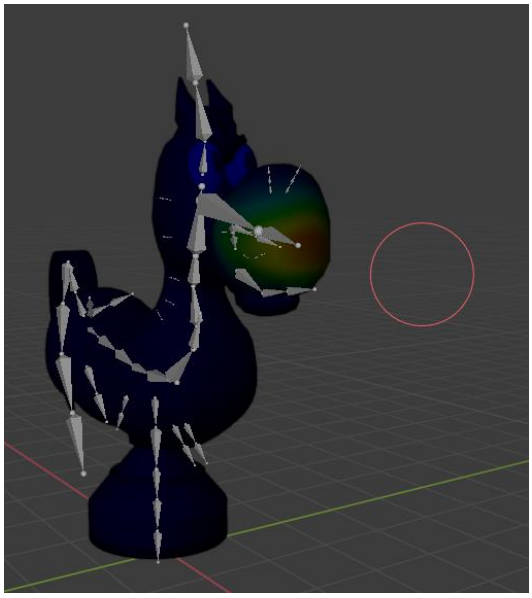

Figura 20 Modelo del caballo en Blender en el modo "Weight Paint". Fuente propia.

- Se pueden aplicar las transformaciones básicas (posición, rotación y escala) a los huesos para grabar las animaciones per se. Blender proporciona un editor en el que se puede indicar la duración de la animación en *frames* y la transformación de los huesos en cada fotograma. Si se pulsa el botón de "*play*" reproduce la animación y se puede ver cómo queda. También se puede cambiar el *frame rate* para aumentar o disminuir la velocidad de la animación.

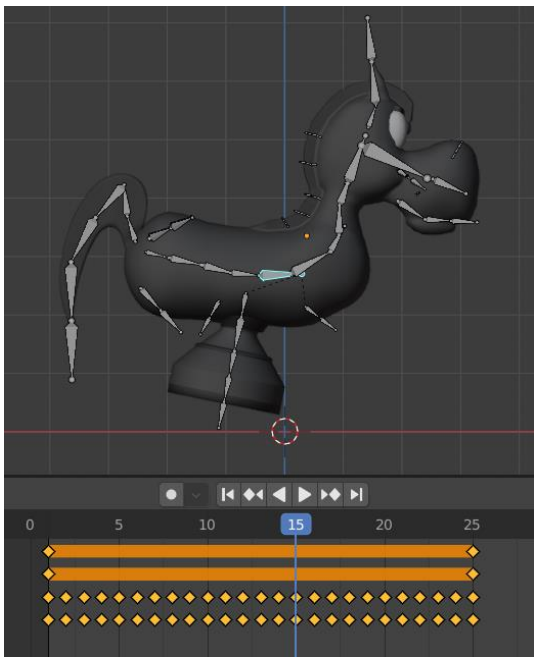

Figura 21 Timeline de una animación del caballo en Blender. Fuente propia.

Una vez se termina de hacer el modelo y las animaciones, simplemente hay que seleccionar el esqueleto y la malla y exportarlo con la extensión *fbx* para que *Unity* lo detecte correctamente. Después solo hay que arrastrar el archivo a la carpeta del repositorio que se quiera y ya podrá ser utilizado dentro del juego, aunque para poder utilizar las animaciones habrá que duplicarlas para tenerlas fuera del *fbx* y que sea posible utilizarlas en el *Animator*[41].

4.2.2.1 Caballo básico

El caballo está formado por el animal y el jinete. Para el caballero se utilizó una lanza (diseñada por Pablo Moreno Martínez) y se creó un escudo que le servirán para atacar y protegerse respectivamente.

Figura 22 Captura del caballo dentro del juego. Fuente propia.

El comportamiento de esta pieza se parece al peón en cuanto a patrulla y persecución, pero el ataque es totalmente diferente. Con tal de simular su movimiento característico en forma de L del ajedrez se propuso que cuando estuviese cerca del jugador saltara encima de él como si de un pisotón se tratase. Cuando el caballo vea al jugador empezará una embestida hacia él, aumentando drásticamente su velocidad y siendo difícil escapar del enemigo. Si consigue escapar, el enemigo volverá a patrullar, pero si llega a acercarse lo suficiente, el caballo saltará encima del jugador. Por lo tanto, si Pawn está justo debajo del caballo, recibirá daño.

[41] Animator https://docs.unity3d.com/es/530/Manual/class-Animator.html

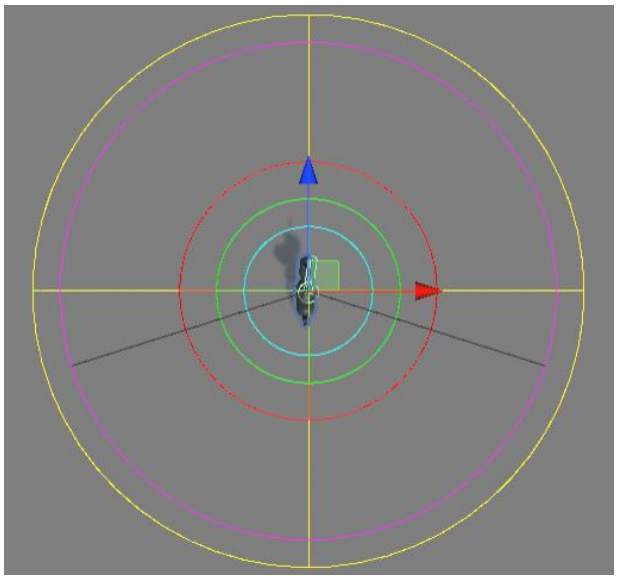

Figura 23 Vista superior de los rangos del caballo. Fuente propia.

La Figura 23 muestra los rangos de detección del caballo. Los que son de color amarillo y magenta tienen la misma función que en el caso del peón, mientras que la circunferencia de color cian indica el rango de defensa y la de color rojo es el rango de salto.

Una vez el caballo toca el suelo pueden pasar varias cosas dependiendo de dónde se encuentra el jugador:

- Si el jugador está fuera del rango de salto, el enemigo seguirá persiguiéndole.
- Si el jugador está dentro del rango de salto, el enemigo volverá a saltar encima de él.
- Si el jugador se queda dentro del rango de defensa, éste se tomará un breve descanso antes de ponerse en posición defensiva. Es en este momento cuando el caballo será vulnerable y podrá recibir daño, pero si ataca y el jinete está defendiendo, empujará al jugador y le atacará con la lanza.

Al igual que en las piezas anteriores, todas las variables que componen al caballo son parametrizables, por lo que se pueden modificar para obtener distintos resultados.

Esta pieza se divide en dos modelos, caballo y jinete, por lo que fue necesario crear dos *Animator* distintos para sus respectivas animaciones. Las animaciones que se crearon para este enemigo fueron:

- **Caballo**: Animación de inactividad, animación de caminar, animación de correr, animación de recibir daño y animación de morir.

- **Jinete**: Animación de inactividad, animación de caminar, animación de correr, animación de recibir daño, animación de morir, animación de defensa y animación de ataque.

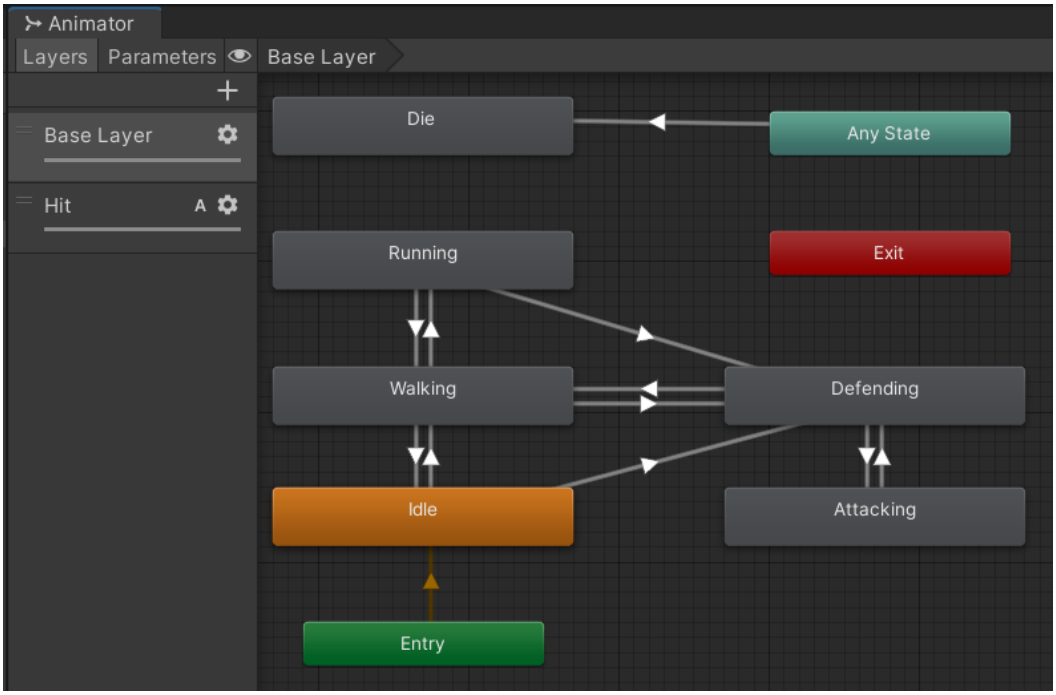

Figura 24 Capa principal del Animator del jinete. Fuente propia.

La Figura 24 muestra la representación de los estados del jinete mediante animaciones. Como se puede observar, el *Animator* puede tener distintas capas; esto sirve para poder mezclar animaciones que deberían poder reproducirse simultáneamente. En este caso, la animación para ser golpeado está separada de la capa principal, ya que puede ser invocada en cualquier momento sin la necesidad de sobrescribir otra. Se puede modificar el peso que tiene cada capa para indicar la influencia que tienen en su mezcla, como muestra la Figura 25.

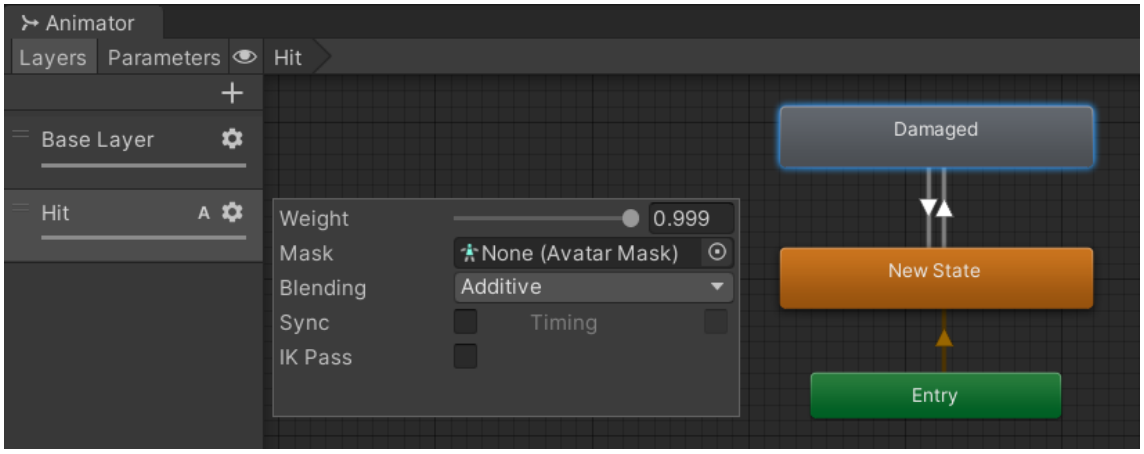

Figura 25 Capa secundaria para la animación de recibir daño del jinete. Fuente propia.

4.2.2.2 Caballo jefe

Se trata de la versión más poderosa de estas piezas. Para su diseño se aprovecharon los modelos ya hechos del caballo básico, simplemente se cambiaron algunos materiales para darle toques dorados con tal de seguir una coherencia artística y narrativa con el peón jefe. También se le añadió un casco al jinete[42].

Figura 26 Caballo jefe dentro del juego. Fuente propia.

Para el escenario se creó una especie de coliseo en Blender y se importó a *Unity*. La zona de combate consistiría en una arena rodeada por un muro decorado con vallas[43] y cuatro paredes de pinchos. Posteriormente, fue añadido en una escena y se añadieron otros elementos decorativos como el terreno, la iluminación y figuras de peones como espectadores, además de elementos del HUD (*heads-up display*) como la barra de vida del enemigo o el menú.

Figura 27 Escena final del nivel 2 con el coliseo. Fuente propia.

[42] Warrior Helmet modelo 3d https://free3d.com/3d-model/warrior-helmet-628685.html
[43] Realistic Fences Pack https://assetstore.unity.com/packages/3d/environments/realistic-fences-pack-211850

El combate contra este oponente se divide en dos fases:

- **1ª fase**: El jefe hará embestidas rápidas en la dirección del jugador y no se detendrá hasta chocarse con la pared del escenario, donde se girará para realizar otra carga. Si choca con el jugador le infligirá daño y el golpe le empujará hacia un lado. Si al final de la embestida el jugador está cerca del enemigo, este se pondrá en posición defensiva, y al igual que con el caballo básico, si Pawn le ataca, el jefe lo empujará y le atacará con la lanza.
  Cada cierto número de ataques de este tipo, el caballo jefe pasará a dar saltos encima del jugador, de forma más agresiva y rápida que la pieza base y sin detenerse. Tras varios saltos volverá a embestir.
  Para poder infligir daño al enemigo en esta fase, el jugador tendrá que provocarlo en la dirección correcta para que se choque con los pinchos que se encontrarán en algunas de las paredes del escenario. En este momento el caballo se aturdirá y Pawn podrá arremeter contra él.

Figura 28 Caballo jefe chocando con los pinchos del escenario. Fuente propia.

- **2ª fase**: Cuando su vida se reduzca a la mitad, el caballo saltará al centro del escenario y soltará su lanza, la cual empezará a girar sobre sí misma y a perseguir al jugador. Al mismo tiempo, aparecerán unos pinchos en la superficie del suelo y el enemigo empezará a perseguir a Pawn. Para superar esta fase el jugador tendrá que acercarse lo suficiente al jefe para que éste salte encima de él, y mientras realice el salto deberá situarse cerca de los pinchos del suelo o saltar por encima de ellos para que cuando el enemigo descienda caiga encima de ellos. Al igual que en la fase anterior, en este momento será vulnerable durante un momento y podrá recibir daño. Deberá

repetir este proceso varias veces a la vez que intenta burlar la lanza, ya que será más rápida que el caballo.

Figura 29 Segunda fase del combate contra el caballo jefe. Fuente propia.

Para este enemigo solo fue necesario crear una animación nueva, la de soltar la lanza, puesto que todas las otras se pudieron reutilizar del caballo básico al compartir los mismos esqueletos.

4.2.3 Alfiles

Los alfiles empezarían a aparecer en el nivel tres del modo historia si este se hubiese desarrollado, pero en este trabajo se crearon para su uso en el modo online, por lo que no fue necesario hacer un jefe de zona.

Puesto que los enemigos anteriores tenían una actitud más agresiva, los alfiles se plantearon como unos enemigos más defensivos o huidizos. Así surgieron dos versiones de esta pieza: alfil rojo y verde.

En cuanto a apariencia se refiere son casi idénticos: ambos son una especie de sacerdote, solo que uno lleva una mitra y un báculo de color rojo y el otro de color verde. Para su diseño ya se contaba con un modelo hecho por Pablo Moreno Martínez, por lo que únicamente fue necesario cambiar los materiales de su arma y sus ropajes en Blender.

Figura 30 Alfil verde y alfil rojo en el juego. Fuente propia.

4.2.3.1 Alfil rojo

El alfil rojo es un enemigo a distancia, esto quiere decir que no tendrá que estar muy cerca del jugador para atacar, sino que le bastará con encontrarse a cierta distancia. Respecto a su comportamiento, patrullará de la misma forma que los anteriores, pero cuando vea al jugador reaccionará de manera diferente dependiendo de la distancia entre los dos.

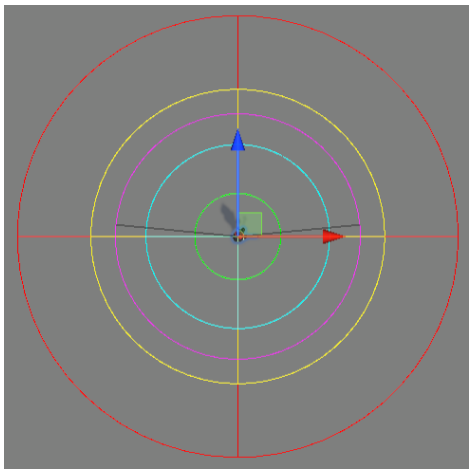

Figura 31 Vista superior de los rangos del alfil rojo. Fuente propia.

Como se puede ver en la Figura 31, los rangos del alfil rojo y sus radios difieren ligeramente del peón básico. La función de estos es la siguiente:

- **Rango de campo de visión y rango de alcance de visión** (magenta y amarillo respectivamente): Tiene el mismo propósito que en casos anteriores, alertar al enemigo de la presencia del jugador. Sin embargo, esta pieza no perseguirá a Pawn mientras esté dentro del alcance de visión, sino que le atacará directamente lanzándole proyectiles.

- **Rango de ataque** (rojo): Si el alfil ve al jugador y se encuentra fuera del rango de alcance de visión, pero dentro del rango de ataque, le seguirá hasta que, o bien se encuentre dentro del alcance de visión, o consiga escapar del rango de ataque. Mientras tanto, seguirá disparando, aunque esté en movimiento.
- **Rango de escape** (cian): Si el enemigo ha visto al jugador y se acerca lo suficiente como para atravesar este rango, el alfil empezará a huir de él en dirección opuesta, aumentando mucho su velocidad. Para derrotarlo habrá que perseguirlo y a la vez esquivar sus ataques, pues no parará de disparar al jugador, aunque esté escapando.

Para el ataque del peón rojo se creó una animación nueva que consistía en levantar el báculo y moverlo antes de lanzar un proyectil.

El proyectil consistía principalmente en un *collider*[44] en forma de esfera que desaparecía al entrar en contacto con cualquier objeto, hiriendo al jugador si se chocaba con él. Para hacerlo visible se usó un sistema de partículas[45] que proyectaba una imagen 2D múltiples veces y que giraba sobre sí misma, generando el efecto visual de una esfera. También se le añadió una luz para que tuviese una mayor integración con el entorno.

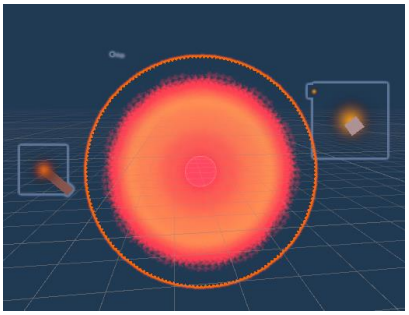

Figura 32 Prefab[46] del proyectil del alfil rojo con sistema de partículas. Fuente propia.

Asimismo, se creó un script para el proyectil cuya principal función era la de perseguir al jugador. Entre sus parámetros se encuentra la velocidad lineal, la velocidad angular, daño y tiempo antes de autodestruirse. Todos estos valores se pueden modificar desde el script de comportamiento del alfil, además del tiempo entre ataques e incluso la posibilidad de atacar con ráfagas (por ejemplo, disparar tres proyectiles seguidos y después esperar un tiempo antes de volver a atacar).

[44] Colliders https://docs.unity3d.com/es/2018.4/Manual/CollidersOverview.html
[45] Sistema de partículas https://docs.unity3d.com/es/530/Manual/ParticleSystems.html
[46] Prefabs https://docs.unity3d.com/es/530/Manual/Prefabs.html

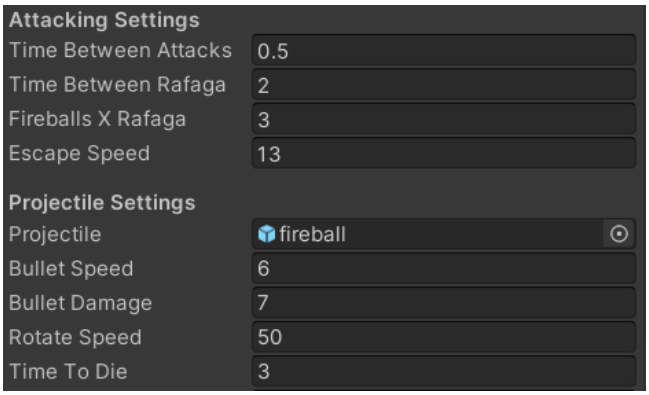

Figura 33 Parámetros del ataque del alfil rojo. Fuente propia.

4.2.3.2 Alfil verde

Por otro lado, el alfil verde es el único enemigo cuya finalidad no es atacar al jugador, sino apoyar a otros enemigos (limitado a los peones). Existen varias vías para explotar esta habilidad dependiendo de los parámetros que se declaren en el *Inspector*, ya que el script permite especificar a qué enemigos curar o si se quiere que el alfil busque enemigos por su cuenta. Las opciones que ofrece este parámetro son:

- **Dos o más objetivos**: El alfil verde patrullará y solo irá a apoyar a un objetivo cuando le quiten vida. Una vez el objetivo recupere toda la vida, el alfil comprobará si alguno de los enemigos a su cargo no tiene el 100% de vida, si es el caso cambiará de objetivo e irá a apoyarlo, pero si todos están bien, volverá a patrullar. Si todos los enemigos de la lista mueren, el alfil pasará a patrullar, y si ha sido indicado así en el *Inspector*, podrá proteger a otro enemigo cuando lo vea, aunque este no estuviese en la lista de objetivos inicial.
- **Un objetivo**: Parecido al caso anterior, pero dado que solamente existe un enemigo al que respaldar, solo le socorrerá a él.
- **Ningún objetivo**: Como se mencionaba en la primera opción, si no tiene objetivos cambiará a modo patrulla, y podrá ayudar a otros enemigos si el parámetro “*Look For Enemies*” está activado, como se muestra en la Figura 34.

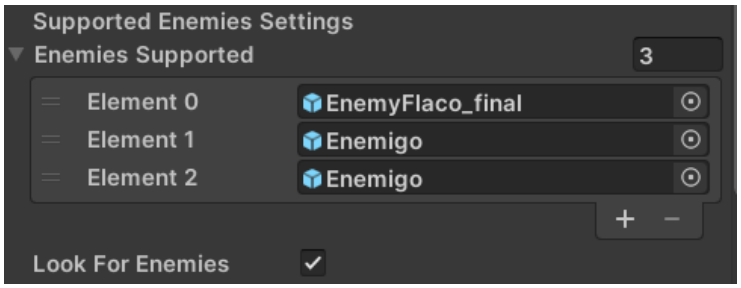

Figura 34 Parámetros del alfil verde para ayudar a enemigos. Fuente propia.

Aunque esta pieza tenga una actitud de soporte, también se defenderá del jugador si lo requiere, actuando de manera diferente dependiendo de su estado en el momento en que vea a Pawn:

- **Tiene uno o más enemigos asignados**: El alfil informará al enemigo con más vida del avistamiento del jugador, es decir, será como si hubiese entrado en el campo de visión del enemigo al que está ayudando. El alfil, mientras tanto, se esconderá a sus espaldas y lo irá curando si recibe daño. Si pasan unos segundos y el jugador ha escapado del alcance de visión del enemigo, dejará de seguirle.
- **No tiene ningún enemigo asignado** (está patrullando): Cuando el jugador se acerque lo suficiente al alfil, éste se quedará quieto mirándole fijamente. Si además atraviesa el rango de ataque, el alfil generará una especie de campo de fuerza a su alrededor que irá aumentando hasta llegar a su máximo. El material del campo de fuerza es un *shader* descargado de un repositorio de GitHub[47]. Si el jugador choca con él recibirá daño y será empujado. Pasado un tiempo, el campo volverá a hacerse pequeño, y el alfil verde esperará un poco para generar otro, momento en el que será vulnerable.

Para este enemigo fue necesario crear una animación nueva en la que levantaba ambos brazos, con tal de simular el ataque cargado del campo de fuerza que aumenta de tamaño. El proyectil, que en este caso servía para curar a los otros enemigos, utilizaba el mismo sistema de partículas que el proyectil del alfil rojo, solo que cambiando los colores rojos por verdes.

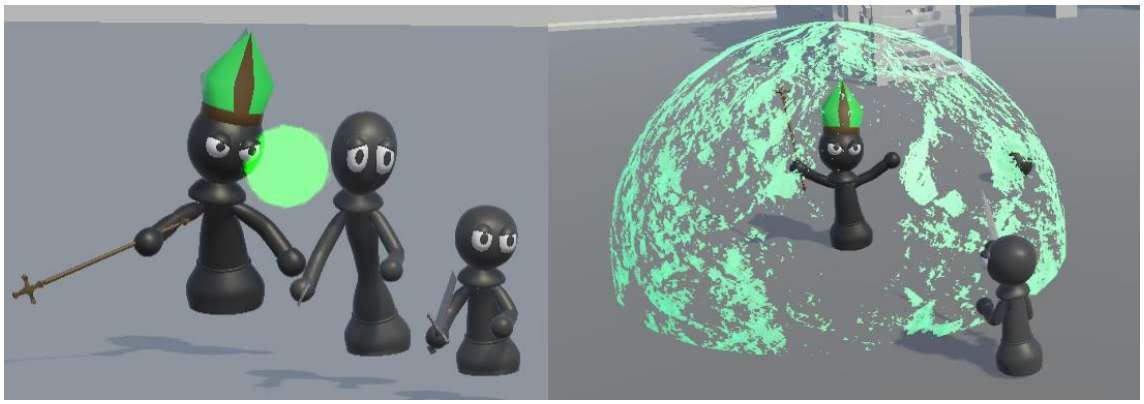

Figura 35 Peón verde apoyando a un enemigo y generando un campo de fuerza. Fuente propia.

[47] Force-Field https://github.com/Brackeys/Force-Field

4.2.4 Torre

A diferencia del resto de enemigos, la torre no es una pieza humanoide, sino más bien una estructura. Está compuesta de tres pisos, cada uno dotado con un cañón y un escalón. Debido a que no se disponía de ningún diseño, fue creada desde cero en Blender.

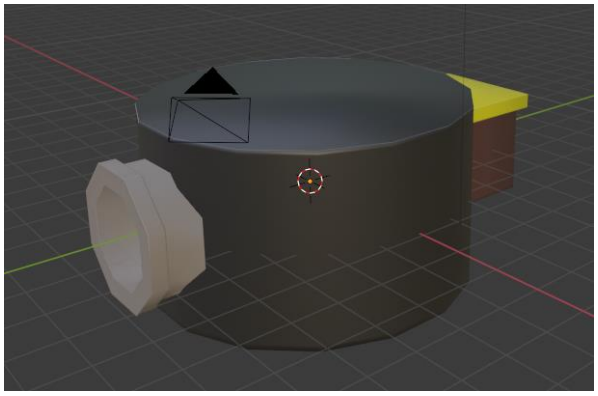

Figura 36 Captura de una de las piezas de la torre en Blender. Fuente propia.

Para poder destruirla habrá que subir los tres escalones y atacar un botón que se encuentra en la parte superior. La dificultad principal reside en que cada planta rotará sobre el eje Y, por lo que el jugador tendrá que calcular bien el salto para no caerse y no ser alcanzado por las bolas de cañón.

Cuando reciba un golpe, la torre expulsará al jugador del piso superior y empezará a perseguirlo durante un tiempo, enfocando sus disparos en él. Después, volverá a su posición inicial y acelerará su velocidad de rotación para que sea más difícil escalarla, pero solo será necesario otro golpe para acabar con ella.

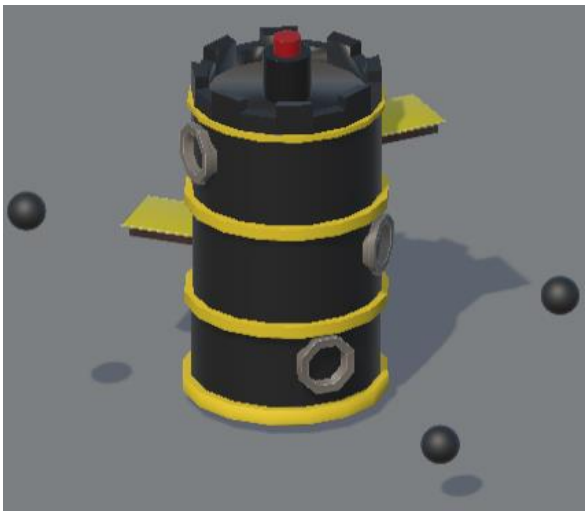

Figura 37 La torre dentro del juego. Fuente propia.

4.2.5 Reina

La reina es el enemigo final del juego, ya que en el modo historia el rey tendría miedo de Pawn y huiría de la batalla. Aunque este nivel no sería implementado para esta parte del desarrollo, se acordó crear la reina para que fuese el enemigo final del modo online realizado por Sergio Pardo Alejandre, con quien se construyó el escenario para este enemigo.

Figura 38 La reina dentro del juego. Fuente propia.

Al tratarse del último enemigo del juego se idearon tres fases para el comportamiento de la reina según su nivel de salud:

- **1ª fase**: La reina lanzaría dagas a gran velocidad hacia el jugador, persiguiéndolo si se alejara de ella. Para infligir daño habría que acercarse a ella, pero al hacerlo cambiaría de ataque y realizaría un movimiento giratorio con su báculo, que se podría esquivar si se salta en el momento justo. Para el proyectil se descargó el modelo de una daga en la página web Turbosquid[48].

Figura 39 La reina lanzando dagas y atacando al jugador en la primera fase. Fuente propia.

[48] Daga Low Poly modelo 3d https://www.turbosquid.com/es/3d-models/dagger-3d-model-1293485

- **2ª fase**: Cuando tenga menos del 66% de vida, la reina se dirigirá al centro del campo de batalla y realizará otro tipo de ataque. En este caso, empezará a girar sobre sí misma y lanzará oleadas de proyectiles (iguales a los del alfil rojo salvo por el color) con tres patrones diferentes que se generarán de manera aleatoria para hacerlo menos predecible. Mientras esté atacando será invencible, ya que la rodeará un campo de fuerza (similar al del alfil verde) que empujará al jugador si se acerca. Para poder dañarla habrá que esperar a que terminen las oleadas y desaparezca el campo de fuerza, momento en el que la reina estará aturdida durante un tiempo antes de volver a atacar.

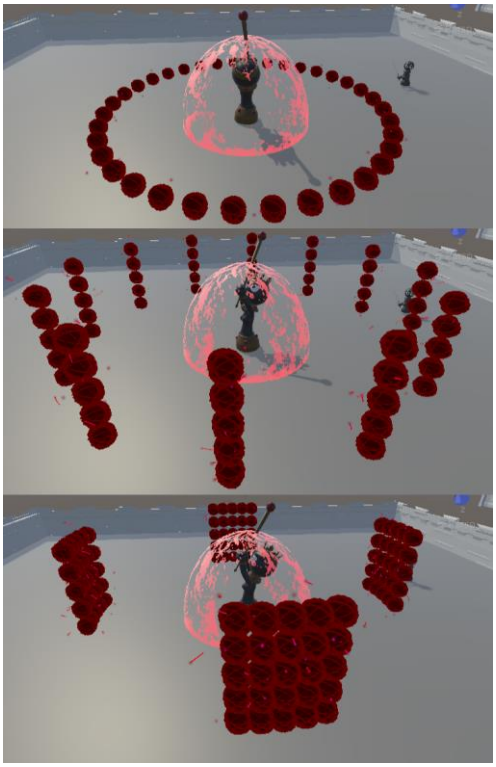

Figura 40 Los tres patrones de ataque en la segunda fase. Fuente propia.

- **3ª fase**: Por último, cuando le quede menos del 33% de vida, la reina se subirá a una especie de balcón[49] que habrá en la sala y comenzará a lanzar rayos sobre el jugador. Estos rayos están compuestos por una luz que indicará la posición donde va a impactar y un sistema de partículas que simula los relámpagos, el cual se obtuvo de la página de Patreon de Hovl Studio[50].
  En esta fase aparecerán del suelo unas columnas[51] que le permitirán al jugador acceder a la posición de la reina. Para llegar al balcón tendrá que ir subiendo

[49] Neoclassical Balcony modelo 3d https://free3d.com/3d-model/neoclassical-balcony-321628.html
[50] Free electricity effects. Made in Unity 2018.1 https://www.patreon.com/posts/21393747
[51] Doric Column modelo 3d https://free3d.com/3d-model/-doric-column--353773.html

por ellos intentando esquivar los rayos. Una vez consiga subir, si asesta un golpe a la reina, el jugador será lanzado otra vez al campo de batalla, por lo que tendrá que volver a subir por las plataformas hasta que consiga derrotarla.

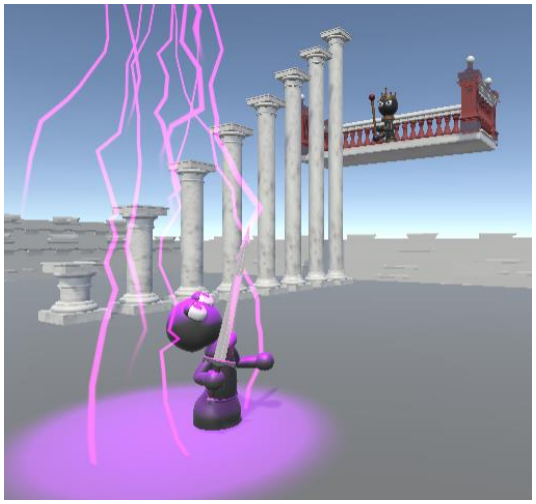

Figura 41 La reina lanzando rayos en la tercera fase. Fuente propia.

El modelo de la reina ya estaba hecho por Pablo Moreno Martínez, junto con las animaciones de caminar e inactividad que fueron heredadas de los anteriores enemigos. Para este enemigo fue necesario crear la animación de lanzar proyectiles (para las dagas y los rayos), el ataque de barrido (cuando el jugador se acerca a ella en la primera fase), el ataque giratorio (cuando gira sobre sí misma en la segunda fase) y la animación de mareo (momento de vulnerabilidad de la segunda fase).

Debido a que tenía un comportamiento más amplio que otros enemigos se decidió usar una funcionalidad del *Animator* que permite crear varias máquinas de estado para dividir las animaciones en diferentes subcapas, una por cada fase, con tal de que fuese más fácil realizar modificaciones si fuera preciso.

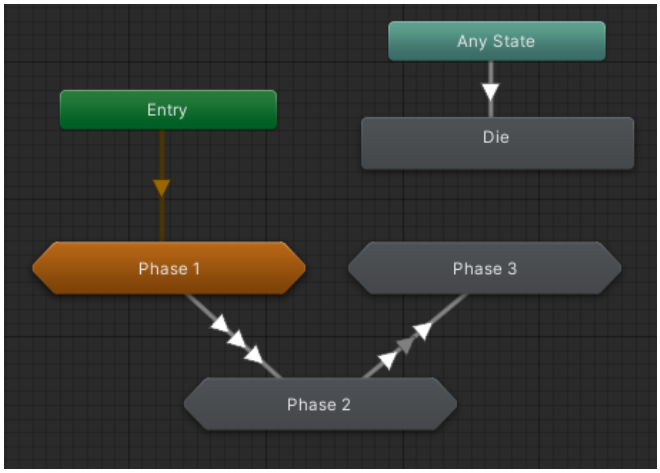

Figura 42 Capa principal del Animator de la reina. Fuente propia.

4.2.6 Parámetros generales

Tras completar todos los enemigos se estandarizaron sus parámetros para que fueran acorde con su lugar en el juego.

La siguiente tabla muestra los datos principales de las piezas creadas, así como si se utilizó GAIA para aprovechar la funcionalidad los árboles de comportamiento (todos menos la torre, ya que solo tenía dos cambios de estado fijos). Para tener una referencia también se ha añadido a Pawn, quien en el nivel 1 tendrá una espada, pero a partir del 2 y en el modo online tendrá una lanza.

	Pawn	**Caballo**	**Caballo jefe**	**Alfil R.**	**Alfil V.**	**Torre**	**Reina**
Vida	100	75	300	75	100	2 golpes	600
Daño	Espada: 20 Lanza: 25	Salto: 15 Lanza: 20	Placaje: 10 Salto: 15 Lanza: 20 Lanza suelta: 5	Bola: 5	Escudo: 10	Bola de cañón: 10	Daga: 10 Cetro: 20 Bola: 5 Rayo: 15
Velocidad de persecución	10 (corriendo)	9	43.5	5	-	5	3.5
Velocidad de huida	-	-	-	13	8	-	-
Rango de visión	-	13.5	-	10	10	-	-
Ángulo de visión	-	215	-	200	200	-	-
GAIA	-	Sí	Sí	Sí	Sí	No	Sí

5 Incorporación de la API GAIA con PandaBT

GAIA es un *asset* creado por José Alapont Luján que permite crear máquinas de estados finitos a partir de ficheros en formato XML. Esta API fue ampliada por Jorge Andreu Royo, quién añadió la posibilidad de usar la aplicación para incluir ficheros que contengan árboles de comportamiento, también en formato XML. Estos ficheros serían *parseados* y añadidos a sus respectivas entidades como un componente de PandaBT.

Así pues, la API proporciona dos funcionalidades diferentes según para qué se quiera usar: máquinas de estado, árboles de comportamiento o ambas. En este TFG se eligió la segunda opción, con tal de comprobar el funcionamiento de la ampliación en un caso real y apoyar el trabajo realizado por Jorge Andreu Royo. Por este motivo, se explicará brevemente el funcionamiento de la versión básica de la API, qué es PandaBT, cómo está integrado en GAIA y su incorporación en Pawn.

5.1.1 GAIA

GAIA proporciona varias plantillas de distintas FSM en formato XML que se pueden modificar para crear máquinas de estado personalizadas. En estas plantillas se especifica el tipo de máquina, así como los estados, las transiciones y sus posibles acciones y eventos, respectivamente.

Para inicializar la API es necesario crear un controlador en la escena, que recogerá los archivos XML. El controlador llamará a otro componente llamado *manager*, el cual ejerce de “unidad central” y gestiona qué máquinas deben ir con qué entidades. Pero primero, llamará a un *parseador*, que se encargará de leer, traducir y, si se da el caso, de imprimir los errores de los archivos XML en un fichero de texto. Por otro lado, se deberá rellenar un fichero “Tags” que contendrá un número entero por cada transición, estado, acción o evento, con sus respectivos nombres. En este fichero también se tendrá que completar un método que, recibiendo una cadena como argumento, devolverá su entero correspondiente que fue asignado anteriormente.

Finalmente, en el método *Start*[52] del script de la entidad que contendrá la máquina se deberá instanciar al *manager*, crear la máquina de estados a partir de él y añadir un evento nulo para indicar que la pila está vacía.

[52] MonoBehaviour.Start() https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html

```
void Start()
{
    FSMmanager = FSM_Controller.INSTANCE.m_managerFSM;
    FSM = FSMmanager.createMachine(this, Tags.CLASSIC, "TankAIDeterministic");
    FSMevents.Add(Tags.UNKNOWN);
}
```

Figura 43 Creación de una FSM por código. Fuente: TFG de Jorge Andreu.

Asimismo, este script deberá contener dos métodos: uno encargado de ejecutar las acciones que recibe de la máquina de estados, que será llamado en el método *Update*, y otro que hará las comprobaciones necesarias para añadir eventos en la máquina de estados, el cual será la función *callback* de la FSM y también se ejecutará en cada fotograma. En el ejemplo de Jorge Andreu estos métodos se llaman *ExecuteAction* y *BuscarEventos* respectivamente.

5.1.2 PandaBT

PandaBT es una herramienta que permite el uso de árboles de comportamiento para la programación de inteligencias artificiales en C#. Está disponible en la Asset Store de *Unity* de manera gratuita, aunque existe una versión de pago que incluye más características (para este desarrollo se optó por la gratuita).

Para su uso es necesario añadir un componente *Panda Behaviour* en la entidad deseada y pasarle como parámetro un fichero de texto que contenga la definición del árbol. La estructura de estos documentos se basa en la declaración de distintos árboles, que parten del hijo nodo del árbol raíz, conocido como *root*.

Según el comportamiento que se persigue, PandaBT proporciona varios nodos con distintas funciones:

- **Tree**: Tiene como parámetro una cadena y puede tener un hijo. Podrá ser referenciado por otros nodos por su nombre. Un *behaviour tree* deberá tener un árbol raíz llamado “*Root*”, el cual será el primero en ser ejecutado.
- **Sequence**: Ejecuta a sus hijos en orden. Devuelve éxito cuando todos sus hijos devuelven éxito, y fracaso cuando uno de sus hijos falla.
- **Fallback**: Ejecuta a sus hijos en orden y seguirá incluso si uno de ellos falla. Devolverá éxito cuando uno de sus hijos devuelva éxito, y fracaso si todos sus hijos fallan.
- **Parallel**: Ejecutará todos sus hijos a la vez. Devuelve éxito cuando todos sus hijos devuelven éxito, y fracaso si alguno de ellos falla.

- **Race**: Ejecutará todos sus hijos a la vez y devolverá éxito cuando uno de ellos devuelva éxito, y fracaso cuando todos fallan.
- **Random**: Selecciona y ejecuta de manera aleatoria uno de sus hijos y devolverá el resultado de éste. Opcionalmente, se puede pasar como parámetros una lista de pesos que establezcan una distribución probabilística para sus hijos.
- **Repeat**: Ejecuta su hijo un número de veces, devolverá éxito siempre que el hijo devuelve éxito en todas las iteraciones, y fracaso si el hijo falla en algún momento. Si no se especifica el número, será infinito.
- **While**: Tiene dos hijos: condición y acción. La acción se ejecuta si la condición devuelve éxito. El nodo devolverá éxito si la acción se ejecuta correctamente y fracaso si uno de sus dos hijos falla.
- **Not**: Devuelve el resultado contrario de su hijo.
- **Mute**: Siempre devuelve éxito cuando su hijo falla o tiene éxito.

Los nodos hoja de los árboles son tareas que se declaran en el código. Para que sean compatibles con PandaBT, será necesario añadir la línea "*using Panda;*", que permitirá al programador usar las funciones de PandaBT. También será necesario escribir la línea "*[Task]" encima* del método respectivo a la tarea. De la misma forma, las variables que se usen como condición tendrán que ser declaradas con "*[Task]"* en la línea superior. Además, debido a que se espera que los nodos devuelvan éxito o fracaso, todas las tareas deberán ser declaradas como *booleanas*.

El componente *Panda Behaviour* permite seleccionar en qué momento lanzar el tic (la señal que recorre el árbol), siendo posible hacerlo en el método *Update*, *LateUpdate*[53], *FixedUpdate*[54] o Manual. Si se marca esta última opción, será necesario llamar al método *PandaBehaviour.Tick()* explícitamente.

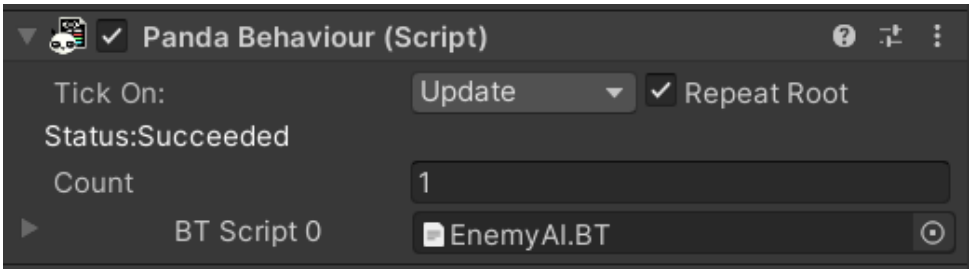

Figura 44 Componente Panda Behaviour en un enemigo. Fuente propia.

[53] MonoBehaviour.LateUpdate() https://docs.unity3d.com/ScriptReference/MonoBehaviour.LateUpdate.html
[54] MonoBehaviour.FixedUpdate() https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html

Una vez se haya creado el fichero *txt* que contenga el árbol y se hayan declarado todas las tareas correctamente en el código, se podrá observar a través del Inspector el recorrido que sigue el árbol en tiempo real cuando se ejecute el juego.

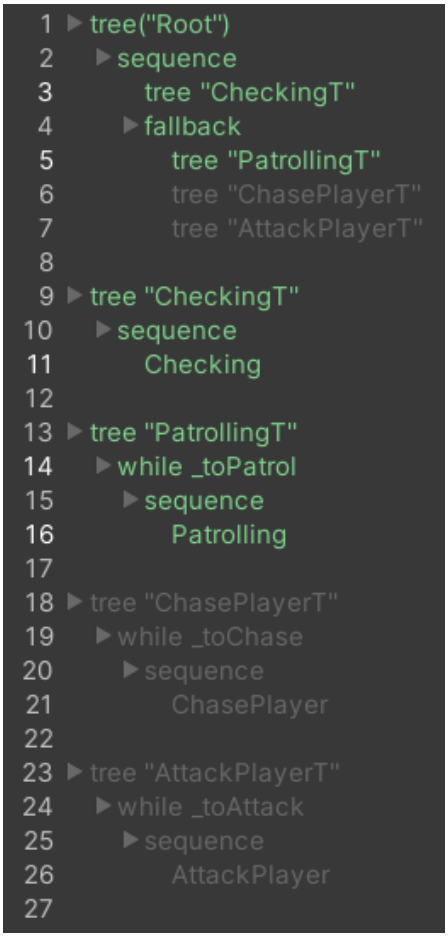

Figura 45 Árbol de comportamiento de PandaBT en tiempo de ejecución. Fuente propia.

5.1.3 Ampliación GAIA con BT

La ampliación de Jorge Andreu Royo consistía en aprovechar los componentes base de GAIA para introducir la compatibilidad con PandaBT. Así pues, la nueva versión permitirá utilizar el controlador no solo para la lectura de ficheros XML que contengan declaraciones de FSM, sino también para árboles de comportamiento en XML, ya que se modificó el *parseador* para que fuese capaz de interpretarlos.

Tampoco será necesario añadir el componente *Panda Behaviour* de manera externa, pues se hará por código en la inicialización del script. Para ello simplemente habrá que crear una instancia del *manager* como se hacía en el caso de las máquinas de estado y posteriormente llamar a la función *createBT(),* que recibe como argumentos la entidad en la que se desea crear el árbol y el ID del árbol, el cual es una cadena

parametrizada en el Inspector y que tiene que ser igual a la etiqueta “BTid” del archivo XML.

```
public void createBT(GameObject character, string bt_id)
{
    try
    {
        PandaBehaviour component = character.AddComponent<PandaBehaviour>();
        component.Compile(BT_dic[bt_id]);
    }
    catch (Exception e)
    {
        Debug.Log("EXCEPTION: " + e);
    }
}
```

Figura 46 Método createBT del GAIA_Manager. Fuente propia.

En resumen, la única diferencia entre crear un árbol usando un componente *Panda Behaviour* y usar GAIA será que el fichero que contenga el árbol tendrá que ser escrito en XML y no en *txt*, y deberá ser añadido al controlador de GAIA y no al componente de PandaBT. El controlador llamará al *manager*, que se encargará de crear el árbol después de pasarlo por el *parseador* y asignarlo a la entidad cuando ésta lo instancie en el método *Start*.

5.1.4 Incorporación en el juego

El primer paso para poder utilizar la herramienta era importarla en el juego. Para ello fue necesario descargarse el repositorio de GitHub que contenía la ampliación de GAIA [55]. Dentro de este proyecto se seleccionó la carpeta que contenía todos los archivos imprescindibles para su uso y se exportó como un paquete.

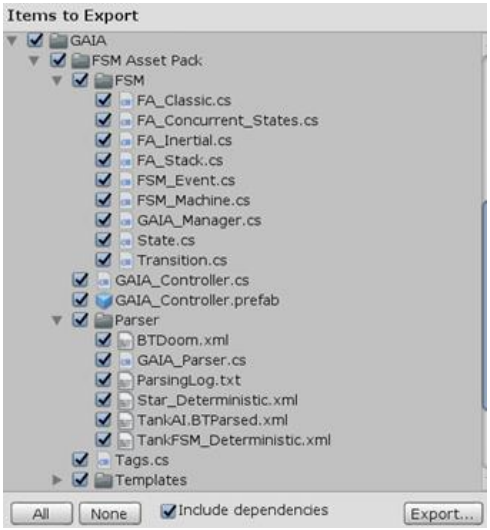

Figura 47 Exportación de los elementos de GAIA desde otro proyecto. Fuente propia.

[55] GAIA https://github.com/tetracube/GAIA

Seguidamente, en el proyecto de Pawn se seleccionó la opción de importar un paquete personalizado y se seleccionó el paquete de *Unity* creado previamente. Esto añadió todos los directorios con sus respectivos archivos en el juego.

El siguiente paso era añadir el *asset* de PandaBT en *Unity*. Para ello se accedió al apartado de Panda BT Free en la Asset Store de *Unity*[56] y se clicó la opción “Añadir a mis assets”. Esta opción abre el *Package Manager* de *Unity*, donde se pueden importar los *assets* descargados en el proyecto actual.

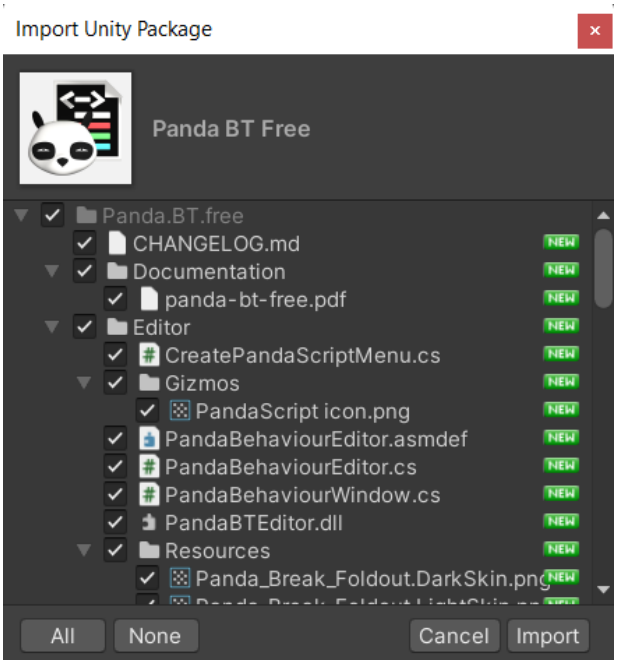

Figura 48 Importación del asset de Panda BT Free. Fuente propia.

A partir de este momento ya se disponía de las herramientas necesarias para poder utilizar GAIA con PandaBT. El proceso para integrar la API en el juego constaba de tres pasos:

- **Modificar el código de los enemigos**: Como se ha explicado en los apartados anteriores, para poder utilizar la API había que realizar una serie de modificaciones en los scripts de los enemigos, empezando por añadir en la cabecera las líneas “*Using GAIA;*” y “*Using Panda;*”. También había que crear una cadena que apareciese como parámetro en el Inspector para indicarle al enemigo el ID del árbol (*BTFileName*), y otra variable de tipo *GAIA_Manager* para instanciar al manager en el método Start y crear el árbol.

[56] Panda BT Free https://assetstore.unity.com/packages/tools/ai/panda-bt-free-33057#publisher

```
        manager = GAIA_Controller.INSTANCE.m_manager;
#if (PANDA)
        manager.createBT(this.gameObject, BTFileName);
#endif
```

Figura 49 Instancia del GAIA_Manager en el método Start. Fuente propia.

Después había que convertir todos los métodos que se pudieran considerar estados a tipo *bool* para que devolvieran éxito o fracaso, así como añadir la línea *"[Task]"* encima de ellos. Lo mismo había que hacer para las variables condicionales.

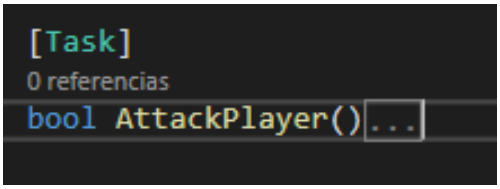

Figura 50 Declaración de un método como una tarea en PandaBT. Fuente propia.

Debido a que el componente *Panda Behaviour* incluye la ejecución del código en cada fotograma al tener el tic por defecto en la opción *Update*, este método ya no sería necesario en los scripts, pero sí sería necesario crear otro idéntico y marcarlo como tarea. Este método, que se decidió nombrar como *Checking*, realizará las comprobaciones necesarias para modificar las variables condicionales que dictarán el camino del árbol. Como se verá en el siguiente paso, será lo primero en ejecutarse.

- **Crear los ficheros XML**: Por cada enemigo que tenga su propio árbol de comportamiento será necesario crear su respectivo archivo XML. La estructura de estos es similar a los *txt* de PandaBT, solo que se usan etiquetas para dividir el árbol. También será necesario asignarle un ID único e indicar para cada árbol si se trata de la raíz.

  A continuación, se muestran dos figuras: la primera corresponde a la equivalencia de la máquina de estados del peón (Figura 17) representada como un árbol de comportamiento; la segunda enseña el árbol raíz del BT de la figura anterior en formato XML, donde se puede ver su ID y los sub-árboles correspondientes al método *Checking*, que actualizará las variables *_toPatrol*, *_toChase* y *_toAttack*, y a los nodos *While*, que representarán el estado actual del enemigo. Tanto el BT completo del peón como los del resto de enemigos están disponibles como anexo.

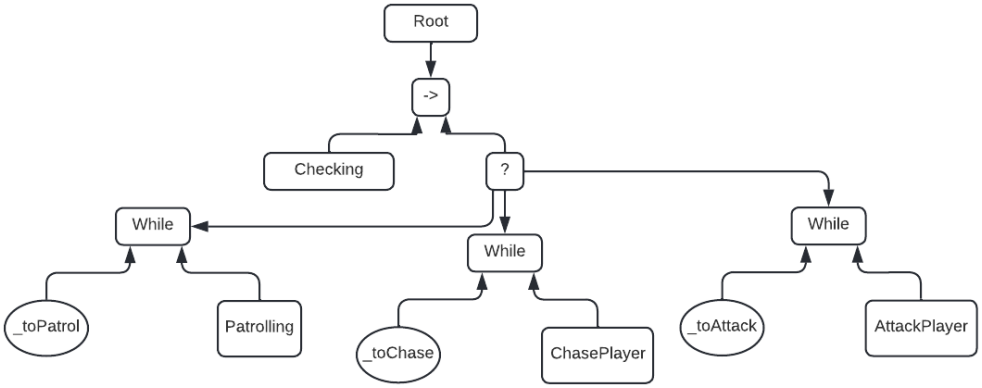

Figura 51 Árbol de comportamiento del peón. Fuente propia.

```
<?xml version="1.0" encoding="utf-8" ?>

<BT>
    <BTid>EnemyAIBT</BTid>
    <Bt>
        <Trees>
            <Tree Root="YES">
                <Name>Root</Name>
                <Child_Nodes>
                    <CN>
                        <CN_Type>node</CN_Type>
                        <CN_Name>sequence</CN_Name>
                        <Child_Nodes>
                            <CN>
                                <CN_Type>tree</CN_Type>
                                <CN_Name>CheckingT</CN_Name>
                            </CN>
                            <CN>
                                <CN_Type>node</CN_Type>
                                <CN_Name>fallback</CN_Name>
                                <Child_Nodes>
                                    <CN>
                                        <CN_Type>tree</CN_Type>
                                        <CN_Name>PatrollingT</CN_Name>
                                    </CN>
                                    <CN>
                                        <CN_Type>tree</CN_Type>
                                        <CN_Name>ChasePlayerT</CN_Name>
                                    </CN>
                                    <CN>
                                        <CN_Type>tree</CN_Type>
                                        <CN_Name>AttackPlayerT</CN_Name>
                                    </CN>
                                </Child_Nodes>
                            </CN>
                        </Child_Nodes>
                    </CN>
                </Child_Nodes>
            </Tree>
```

Figura 52 Árbol raíz del fichero XML que contiene el BT del peón. Fuente propia.

- **Añadir el controlador en la escena**: Para todas las escenas en las que hubiera enemigos será necesario añadir el script *GAIA_Controller* a algún *gameObject* (normalmente al *Game Manager*). Este script tiene dos

parámetros: una lista de ficheros XML para FSM y otra lista de ficheros XML para *behaviour trees*. Puesto que solo se usa la funcionalidad de PandaBT, se podrá ignorar el primer parámetro, mientras que en el otro habrá que incluir todos los archivos XML utilizados por los enemigos en la escena.

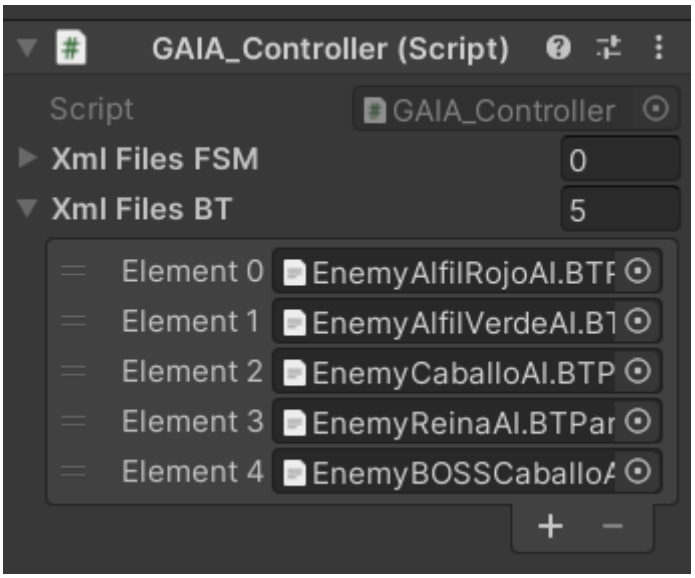

Figura 53 Componente GAIA_Controller con la lista de los archivos XML con los BT de los enemigos en la escena. Fuente propia.

De esta forma, se crearon todos los ficheros XML y se añadió un controlador en todas las escenas en las que ya había enemigos, incorporando así la API de GAIA con PandaBT en el juego.

6 Pruebas realizadas

Al finalizar la programación de los enemigos se comprobó que tanto su comportamiento como sus animaciones funcionaban correctamente.

Aunque todos los enemigos funcionaban como se había planeado, un error en el método encargado de crear el componente *Panda Behaviour* en tiempo de ejecución, *PandaBehaviour.Compile()*, impedía ver el contenido del componente en el *Inspector*, por lo que si se intentaba abrir para ver el árbol en tiempo de ejecución (Figura 45) la consola imprimía una excepción del tipo *NullReferenceException* respectiva al editor de PandaBT.

Ya que no había información específica sobre este error ni en la documentación ni en los foros de PandaBT, se supuso que la excepción *NullReference* se debía a que uno de los parámetros del componente (Figura 44) no se inicializaba correctamente en el *Inspector*, por lo que se decidió hacerlo de forma explícita desde el código. Así pues, se modificó el método *createBT()* de la Figura 46 para que, en caso de estar trabajando en el editor de *Unity* (puesto que no tenía utilidad en la aplicación de escritorio), creara un archivo *txt* a partir del XML *parseado* por GAIA, accesible desde el diccionario *BT_dic*, y se lo añadiera al componente *PandaBehaviour* creado anteriormente. De esta forma también se almacenarían en una carpeta todos los BT en formato de texto. La Figura 54 muestra el resultado final de este método.

```
public void createBT(GameObject character, string bt_id)
{
    try
    {
        PandaBehaviour component = character.AddComponent<PandaBehaviour>();
        component.Compile(BT_dic[bt_id]);

        if (Application.isEditor)
        {
            string temporaryTextFileName = Application.dataPath + "/Resources/BTFilesForGAIA/" + bt_id + ".txt";
            File.WriteAllText(temporaryTextFileName, BT_dic[bt_id]);
            component.scripts = new TextAsset[1];
            TextAsset arbol = Resources.Load(temporaryTextFileName) as TextAsset;
            component.scripts[0] = arbol;
        }

    }
    catch (Exception e)
    {
        Debug.Log("EXCEPTION: " + e);
    }
}
```

Figura 54 Modificación del método createBT. Fuente propia.

También se modificó el archivo *GAIA_Controller* para que creara la carpeta “BTFilesForGAIA” en caso de que no existiese el directorio, siempre y cuando se encontrara también en el editor y no en la aplicación, como muestra Figura 55.

```
#if (PANDA)
        //Loads and parses all xml definitions of BTs
        if (m_xmlFilesBT != null)
        {
            string curDirectory = Application.dataPath + "/Resources/BTFilesForGAIA";
            if (!Directory.Exists(curDirectory) && Application.isEditor)
            {
                var folder = Directory.CreateDirectory(curDirectory);

            }
            for (int i = 0; i < m_xmlFilesBT.Length; i++)
            {
                //Debug.Log("PARSEANDO FICHEROS PANDA");
                m_manager.addBT(m_xmlFilesBT[i].text);
            }
        }
#endif
```

Figura 55 Parte del método Awake[57] del archivo GAIA_Controller encargada de enviar los BT en formato XML al manager. Fuente propia.

Tras estas modificaciones ya se pudo observar los árboles de comportamiento en tiempo de ejecución, y se obtuvo la utilidad extra de guardar en una carpeta aparte los archivos XML en formato *txt*.

El siguiente paso era mover los enemigos desde la zona de pruebas donde habían sido creados y testeados hasta el nivel 2 y el modo online.

El primer caso no supuso ninguna dificultad debido a que era un proceso parecido al realizado en el primer nivel: añadir los *prefabs* de los enemigos en el mapa, asignarles puntos de patrulla si se deseaba, hacer el *bake* del *NavMesh Surface* y asegurarse de que el *GAIA_Controller* estaba en la escena.

Figura 56 Nivel 2 con los enemigos añadidos. Fuente propia.

Para el segundo caso, sin embargo, fue necesario modificar los enemigos para que en todo momento tuviesen las referencias de los jugadores actuales en la sala,

[57] MonoBehaviour.Awake() https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html

actualizando su objetivo al que más cerca estuviese de ellos. También hubo que modificar el entorno, ya que los enemigos aparecían en zonas aleatorias y podrían acceder a sitios de los que luego no podían salir.

Sergio Muñoz se encargó de duplicar los *prefabs* de los enemigos para crear unas versiones nuevas para el multijugador. Estas versiones hacían uso de la herramienta *Photon*[58], utilizada para implementar el modo online. Los componentes de este *framework* serían los encargados de sincronizar las animaciones, las posiciones y las rotaciones de los enemigos. Lamentablemente, existían problemas en la sincronización de la vida de los enemigos que hicieron que en algunas partidas se experimentaran incoherencias en las animaciones o en las fases de la reina.

Figura 57 Modo online con los enemigos. Fuente propia.

Como muestra Figura 57, el modo online consiste en una serie de oleadas en las que los jugadores deberán derrotar a todos los enemigos para pasar a la siguiente. Cada oleada se centrará en un tipo de enemigo, y cuando se superen todas se accederá a una sala con la reina (Figura 58).

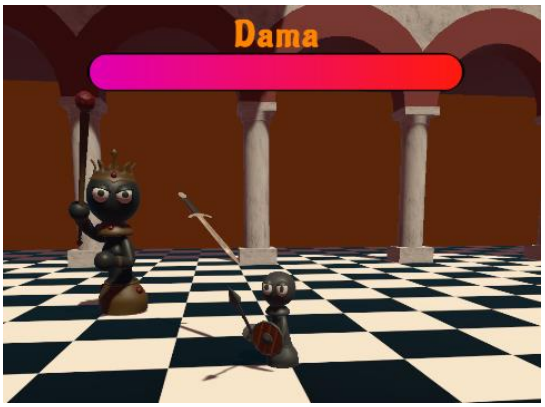

Figura 58 Sala de la reina en el modo online. Fuente propia.

[58] Photon https://www.photonengine.com/pun

Por último, cuando se creó el ejecutable del juego se descubrió un error de GAIA relacionado con los directorios que impedía que los enemigos funcionasen correctamente. Esto se debía a que intentaba acceder a una carpeta para sobrescribir el fichero *ParsingLog*. Sin embargo, esta carpeta solo estaba disponible en el editor, y no en el directorio del ejecutable, por lo que el fallo impedía que se crearan los componentes de PandaBT en los enemigos en la aplicación de escritorio.

```
public GAIA_Parser()
{
    LoadedMachines = LoadedSubMachines = LoadedBTs = numErrors = numErrorsBT = 0;
    string logPath = System.IO.Directory.GetCurrentDirectory() + "/Assets/GAIA/FSM Asset Pack/Parser/ParsingLog.txt";
    file = new System.IO.StreamWriter(logPath);
}
```

Figura 59 Versión original del script GAIA_Parser

Este error se solucionó añadiendo unas líneas de código en el *GAIA_Parser* para filtrar las excepciones. Si no se encontraba el archivo en la primera carpeta significaría que no está en el editor, por lo que intentaría acceder al directorio correcto en la localización del ejecutable. Si tampoco lo encontraba, significaría que el archivo todavía no existe, por lo que lo crearía desde el código. Este último caso se daba la primera vez que se ejecutaba el juego o si simplemente el archivo había sido eliminado.

En la Figura 59 se muestra la primera versión de *GAIA_Parser*, y en la Figura 60 la modificación para el correcto funcionamiento de la aplicación.

```
public GAIA_Parser()
{
    LoadedMachines = LoadedSubMachines = LoadedBTs = numErrors = numErrorsBT = 0;
    string logPath = System.IO.Directory.GetCurrentDirectory() + "/Assets/GAIA/FSM Asset Pack/Parser/ParsingLog.txt";

    try
    {
        file = new System.IO.StreamWriter(logPath);

    }
    catch (Exception)
    {
        logPath = System.IO.Directory.GetCurrentDirectory() + "/ParsingLog.txt";

        try
        {
            file = new System.IO.StreamWriter(logPath);
        }
        catch (Exception)
        {
            logPath = System.IO.Directory.GetCurrentDirectory() + "/ParsingLog.txt";
            using (StreamWriter sw = File.CreateText(logPath)) ;
            file = new System.IO.StreamWriter(logPath);
        }
    }
}
```

Figura 60 Código que resuelve un problema con los directorios en GAIA. Fuente propia.

7 Conclusiones

Los objetivos que se plantearon al principio del desarrollo de este TFG se alcanzaron con éxito: se creó la escena final del nivel 2, se crearon los modelados y animaciones necesarias, se diseñaron todos los enemigos previstos y se les añadió las utilidades de GAIA con PandaBT.

El resultado ha sido mejor de lo esperado, sobre todo en el apartado de animación y modelado, pero también en cuanto a programación. En la asignatura D3D se desarrollaron tan solo dos enemigos con un comportamiento simple, mientras que en este proyecto se crearon seis con una conducta más amplia y una implementación mucho más compleja.

La colaboración con el resto de los compañeros ha supuesto una carga de trabajo extra en cuanto reuniones y coordinación se refiere, pero en ningún caso un inconveniente. Hacer el TFG de un videojuego de manera conjunta ha sido un proceso que ha requerido mucho esfuerzo por parte de los tres componentes, especialmente en el aprendizaje de nuevas áreas de las que no se tenía conocimiento, como ha sido la animación 3D en este caso.

Por lo que respecta a GAIA, ha demostrado ser útil para la centralización de los ficheros XML con los BT de los enemigos, por lo que la recomendaría para futuros desarrollos siempre y cuando se trate de un proyecto lo suficientemente grande como para que la incorporación de esta herramienta no suponga una inversión de tiempo demasiado grande.

8 Relación con los estudios cursados

Un gran porcentaje del proyecto ha consistido en programar, por lo que asignaturas como "*Programación*", "*Estructuras de Datos y Algoritmos*" y "*Competición de programación*" han sido imprescindibles para tener la soltura y los conocimientos necesarios en lenguajes orientados a objetos.

Al tratarse de un TFG colaborativo, también influyeron asignaturas como "*Gestión de Proyectos*" o "*Ingeniería del Software*". En ISW se empezó a trabajar con GitHub para el manejo de un proyecto y el control de versiones, mientras que de GPR se han utilizado métodos de organización temporal como el diagrama de Gantt.

En cuanto a mi rama de especialización, computación, ha destacado sobre todo la asignatura de "*Agentes Inteligentes*". Esta supuso el primer contacto de la carrera en simulación de agentes capaces de interactuar con el entorno, y también se aprendieron técnicas de comunicación entre agentes, lo que ha servido de gran ayuda para el diseño de los enemigos. "*Algorítmica*" también ha tenido un papel importante en el aspecto técnico del juego y la generación de código óptimo, y "*Sistemas Gráficos Interactivos*" ha servido tanto para obtener agilidad en el uso de operaciones vectoriales (muy abundantes en los juegos) como para la comprensión de la estructura de los *shaders* y otros apartados gráficos del juego.

Por otro lado, "*Técnicas, entornos y aplicaciones de inteligencia artificial*" y "*Aprendizaje Automático*" permitieron tener una visión general del campo de la inteligencia artificial, así como sus diferentes aplicaciones en la actualidad.

La familiaridad adquirida con *Unity* viene de asignaturas optativas del bloque de videojuegos, como "*Desarrollo de videojuegos 3D*" y "*Desarrollo de videojuegos 2D*", siendo la primera el nacimiento de este trabajo. Dentro de este bloque también se encontraba "*Animation and design of videogames*", que ayudó a tener una base práctica en la creación de máquinas de estados de animación, sistemas de partículas y cinemáticas. Aunque se encuentra fuera de este bloque, "*Interfaces persona computador*" también ayudó en la gestión de interfaces y elementos del HUD.

Por último, cabe destacar asignaturas como "*Deontología y profesionalismo*" e "*Inglés intermedio bajo para la informática*", que, si bien no han formado parte directa en el desarrollo práctico, han demostrado ser útiles para la correcta lectura e interpretación de artículos legislativos o artículos científicos en inglés.

9 Trabajo futuro

Por lo que al juego se refiere, si se quisiera llevar adelante sería necesario pulir y limpiar todo lo que se ha desarrollado hasta el momento antes de seguir avanzando. Al encargarse de tareas totalmente diferentes (aunque en un espacio de trabajo común), se añadió una gran cantidad de contenido al proyecto por parte de los tres componentes del grupo que no ha terminado por usarse al 100%. Esto tiene varios inconvenientes, y es que dificulta la navegación por los directorios del proyecto durante el desarrollo y puede aumentar significativamente el tamaño del ejecutable final.

Posteriormente, habría que preguntarse si la continuidad del desarrollo sería con fines personales o comerciales. El primer caso tendría como consecuencia un desarrollo más desinteresado y menos lineal, aunque en caso de finalizarlo sería un producto que mostraría las capacidades de los desarrolladores y serviría de posible carta de presentación para las compañías. El segundo caso podría implicar un rediseño del concepto y también plantear otras cuestiones como la creación de una empresa, la contratación de personal o la búsqueda de inversores o subvenciones. De todas formas, el desarrollo de ambas opciones tendría un camino similar: completar el alcance planteado en el GDD.

En cuanto a la API de GAIA, sería interesante ampliarla una vez más de tal forma que se pudiesen crear árboles de forma gráfica, una de las funciones que permiten otros *assets* de árboles de comportamiento. A partir de este editor, por ejemplo, se podrían generar los XML de manera automática para después usarlos en GAIA.

10 Bibliografía

[1] Alapont Luján, J. (2020, 11 noviembre). API de gestión de Inteligencia Artificial basada en las máquinas de estados finitos en C#. RiuNet repositorio UPV. https://riunet.upv.es/handle/10251/47429

[2] Andreu Royo, J. (2021, 21 abril). Ampliación del Asset Game Artificial Intelligence para Unity incorporando Behavior Trees. RiuNet repositorio UPV. https://riunet.upv.es/handle/10251/165380

[3] Colledanchise, M., Parasuraman, R., & Ogren, P. (2019). Learning of Behavior Trees for Autonomous Agents. IEEE Transactions on Games, 11(2), 183–189. https://doi.org/10.1109/tg.2018.2816806

[4] Gajardo, I., Besoain, F., & Barriga, N. A. (2019). Introduction to Behavior Algorithms for Fighting Games. 2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON). https://doi.org/10.1109/chilecon47746.2019.8988008

[5] Zutell, J. M. (2022, 10 marzo). Flexible Behavior Trees: In search of the mythical HFSMBTH for Collaborative Autonomy in Robotics arXiv.Org. https://arxiv.org/abs/2203.05389

Anexo I: GDD

1. Datos principales

Título: “Pawn”.

Concepto: Eres Pawn, un peón al servicio de sus reyes. Cansado de las injusticias y la guerra constante a la que eres sometido, decides empezar tu propia guerra, esta vez, contra tu ejército, y así derrocar al rey con tus propias manos.

Tema: Ajedrez, mundo distópico medieval.

Género: Plataforma 3D, arcade, aventura.

Plataforma: PC, Android, Realidad virtual para dispositivos Android.

Mercado: Todos los públicos.

PEGI: Edad mínima recomendada: 7 años. Esto es debido a que, pese a no haber violencia explícita, sangre o palabras malsonantes, sigue siendo un juego cuyo objetivo es matar, aunque sean figuras de ajedrez.

2. Pitch doc

2.1 Resumen

En el siglo VI d.C. estalló una guerra entre dos bandos. Las piezas blancas intentaban conquistar el territorio de las piezas negras. Pawn, un peón negro, está harto de tanta violencia y de ser utilizado por el Rey a su conveniencia. Un día empieza a cuestionarse por qué pelea y llega a la conclusión de que la disputa ya no tiene sentido. Tanta es su determinación que incluso decide rebelarse y acabar con la guerra cueste lo que cueste. Incluso si ha de matar a su Rey con sus propias manos.

2.2 Gameplay

El juego empieza manejando a Pawn, concretamente dentro de una jaula de la que habrá que escapar. Tras las puertas de esta se encuentra un camino lleno de enemigos y obstáculos que habrá que superar para poder avanzar hacia el objetivo inicial, el castillo del Rey.

Será en tercera persona y para moverse se usarán las teclas de dirección (WASD), además, se podrá saltar (espacio), atacar (botón izquierdo del ratón), esquivar (Q) y bloquear (botón derecho del ratón).

Cada nivel tiene al final un jefe representativo del nivel, siendo correspondiente a la jerarquía del ajedrez: Peón, Caballo, Alfil, Torre, y Dama.

De igual forma, cada nivel tendrá un escenario diferente, donde se encontrará a enemigos de niveles inferiores, además de los del propio nivel.

Al acabar con el jefe de cada nivel se recibirá un objeto o habilidad como recompensa.

2.3 Estética

El juego está ambientado en los campos de batalla medievales, donde el objetivo de cada ejército era conquistar el castillo enemigo.

El estilo escogido es *cartoon low poly*, con colores variados y brillantes, basados en una paleta de colores concreta. Es importante recalcar que Pawn es un peón negro, al igual que todos los enemigos a los que se enfrentará, por ello es importante tener en cuenta los colores que podrían contrastar en la ambientación

Los personajes son figuras representativas del ajedrez caricaturizadas.

3. Análisis competitivo

3.1 Referentes

A nivel jugable: *Crash Bandicoot, Spyro, Ratchet & Clank.*

Ilustración Anexo 1 Captura del videojuego Crash Bandicoot

A nivel estético: *Fall guys, Human: Fall Flat.*

Ilustración Anexo 2 Imagen promocional de Fall Guys

3.2 Competencia

Además de los juegos nombrados con anterioridad, a nivel de jugabilidad podemos destacar también los siguientes títulos: *Super Mario (64, Galaxy, 3D World), Sonic Generations, Jak and Daxter*. Esto sin tener en cuenta los juegos de *plataforma 2D*, de los cuales hay una gran variedad actual.

Ilustración Anexo 3 Captura del juego de Jack and Daxter

Ilustración Anexo 4 Captura del juego de Super Mario Odyssey

3.3 Características diferenciadoras:

Estilo artístico *low poly*, figuras caricaturescas.

Los personajes están basados en las figuras del ajedrez. Protagonizado por un peón que, por una serie de sucesos, se rebela contra el sistema soberano. Una referencia clara es la novela de George Orwell, *1984*.

Además, no se ha podido encontrar ningún juego en el que los protagonistas sean figuras de ajedrez, sin contar el ajedrez en sí.

4. Estructura Narrativa

4.1 Deseo principal:

El deseo principal del protagonista, Pawn, es avanzar entre las líneas enemigas (antes aliadas) con el objetivo de derrocar al Rey y vencer a todo su ejército.

4.2 Actos narrativos:

Inicio: Pawn comienza con un cuento ilustrado en las páginas de un libro que dará contexto de la historia principal del videojuego.

Primer nivel: Pawn se encuentra en medio de un campo de batalla con el objetivo de derrotar a todos los enemigos del nivel para así avanzar hasta la caballería.

Cuando se derroten a todos, en la villa del final aparecerá un peón gigante aparecerá con el que tendremos que pelear. Este enemigo corresponde al jefe del primer nivel.

Segundo nivel: Tras su derrota, se pasará al siguiente nivel, con una lanza como arma.

Este nivel está dividido en dos partes. La primera será similar al nivel 1, pero añadiendo nuevos enemigos (los caballos); la segunda corresponde a un modo de juego que se denomina *endless runner,* basado en correr hacia delante de manera automática, sin parar, pudiendo solo moverse hacia los lados y saltar. Si se colisiona con un obstáculo, quitará vida al personaje. Con tres golpes está muerto.

Al pasar un minuto este nivel terminará y aparecerá un peón montado a caballo. Este es el jefe de este nivel, el cual correrá hacia el personaje e impactará contra él con su lanza. Tras derrotarlo, se conseguirá un escudo como recompensa.

Tercer nivel: Comienza estando fuera de las murallas que custodian el castillo real. Aquí habrá que vencer a los enemigos que aparezcan en oleadas para detenerlos. Entre los enemigos se podrá encontrar peones, caballos y alfiles. Estos últimos se dividen en dos tipos: alfil verde y alfil rojo. El verde se encarga de curar a otros enemigos o protegerse con un escudo mágico para que no le ataquen; el rojo lanza orbes de energía que harán daño a Pawn mientras retrocede cada vez que se acerca el protagonista.

Tras superar varias oleadas de enemigos mientras se avanza hacia la muralla del castillo, aparecerá el jefe alfil, al derrotarlo se verá una cinemática donde lo lanzaremos con nuestra lanza contra la pared de una de las torres, rompiéndola y creando un agujero que permitirá avanzar al siguiente nivel. El ítem que dejará este *boss* es una mitra obispal que permitirá al jugador realizar un doble salto.

Cuarto nivel: Al adentrarse en la torre a través del hueco surgido en la anterior batalla, deberá subir las escaleras en espiral que hay en la torre, derrotando a los enemigos y saltando los espacios vacíos entre escalones y obstáculos en el camino. Los enemigos que hay son todos los anteriores, además de unos peones subidos en pequeñas torres que disparan cañones que empujan hacia atrás al personaje, pero si se usa el escudo se podrá amortiguar el impacto.

Al llegar a la cima de la torre se encuentra con otra torre que será el jefe de este nivel. Tras acabar con ella se obtendrá un pequeño cañón como arma adicional a distancia.

Quinto nivel: Una vez en la cima de la torre, se tendrá que bajar hasta el patio de armas. Para ello habrá que buscar la manera óptima de hacerlo mediante saltos.

Al llegar al patio de armas habrá que enfrentar a oleadas de enemigos y, entre ellas, aparecerán 3 jefes que son: un caballo, un alfil y una torre. Cada uno dejará como recompensa por derrotarlo una parte de una llave. Al tener las 3 partes, se unirán para poder acceder al castillo real.

Sexto y último nivel: Tras derrotar a los enemigos anteriores, se podrá acceder a las puertas del castillo. Una vez dentro se verá una cinemática donde se encuentran el Rey y la Dama, al percatarse de la presencia del jugador, la Dama apartará al Rey para enfrentarse a él y, de este modo, comenzar la pelea final contra el último jefe del juego.

Desenlace o final del juego: Cuando la Dama es derrotada, se verá una cinemática donde el Rey demuestra ser un cobarde al huir. En su huida deja caer la corona real y se podrá ver cómo

Pawn se acerca y la recoge, acabando el juego con la imagen del protagonista alzando la corona en sus manos, dudando entre acabar con la figura del monarca definitivamente, o simplemente ocupar el trono que acaba de quedar vacío.

5. Guion de la historia principal

Aparece un libro con ilustraciones y un texto en la parte inferior que narrará la historia.

Primera página del libro:

- Se ve una imagen del campo de batalla, con las piezas blancas y negras peleando.

- Narrativa en formato novela visual (el texto se vería en la parte inferior de la pantalla):

Texto 1 → Corría el siglo VI d.C. Las piezas blancas ansiaban más tierras de las que tenían y decidieron invadir el territorio de las piezas negras.

Texto 2 → El bando atacado, como es normal, no aceptó y estalló la guerra.

Texto 3 → La guerra duraría años, e incluso milenios...

Segunda página:

- Se ve una imagen de Pawn.

- Narrativa:

Texto 1 → Esta historia cuenta las aventuras de Pawn, un peón negro sin ninguna habilidad especial que le diferencie de los demás peones.

Texto 2 → Pawn, al contrario que sus compañeros, tenía unos ideales diferentes. No le bastaba con ser una herramienta que pudiera morir con el fin de proteger al Rey.

Texto 3 → Él quería ser más valioso, ser algo más en la vida, que la sociedad le reconociera y le recordaran como una gran figura de ajedrez.

Tercera página:

- Imagen de Pawn dando una charla a sus compañeros.

- Narrativa:

Texto 1 → Harto de ver morir a tantos de los suyos, compañeros y compañeras que habían dado su vida a una causa injusta. Pawn había tomado una decisión…

Texto 2 → Una revolución junto a sus compañeros sería el golpe perfecto para derrocar a su vil rey, pero…

Cuarta página:

- Imagen de Pawn encerrado.

- Narrativa:

Texto 1 → Pero Pawn no contaba con que sus propios aliados lo delataran como traidor y lo encarcelaran por su conspiración contra el Rey.

Quinta página:

- Imagen de Pawn mirando al castillo desde su jaula.

Texto 1 → Decepcionado y aislado de todos, su visión del mundo cambió.

Texto 2 → Él acabaría con la guerra a cualquier precio.

Sexta página:

- Imagen de Pawn mirando una espada apoyada contra la rueda de la jaula en la que se encuentra encerrado.

Texto 1 → Destruiría a todos aquellos que se pusieran en su camino hasta su propio Rey y así arrebatarle la corona para finalizar la guerra de una vez por todas.

Texto 2 → Esa sería su meta en la vida. La partida acababa de comenzar...

Finaliza la narrativa y comienza el juego → Se cierra el libro y se va oscureciendo la pantalla para luego mostrar el primer nivel.

6. Flowchart de la narrativa:

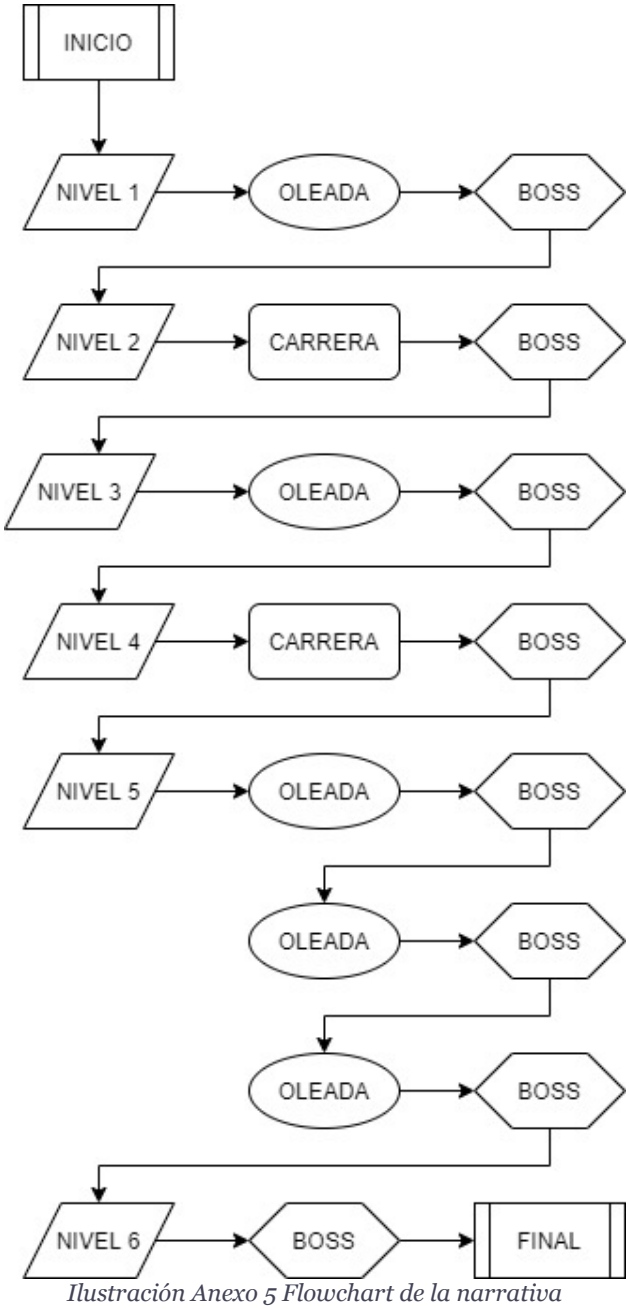

Ilustración Anexo 5 Flowchart de la narrativa

7. Curvas de interés:

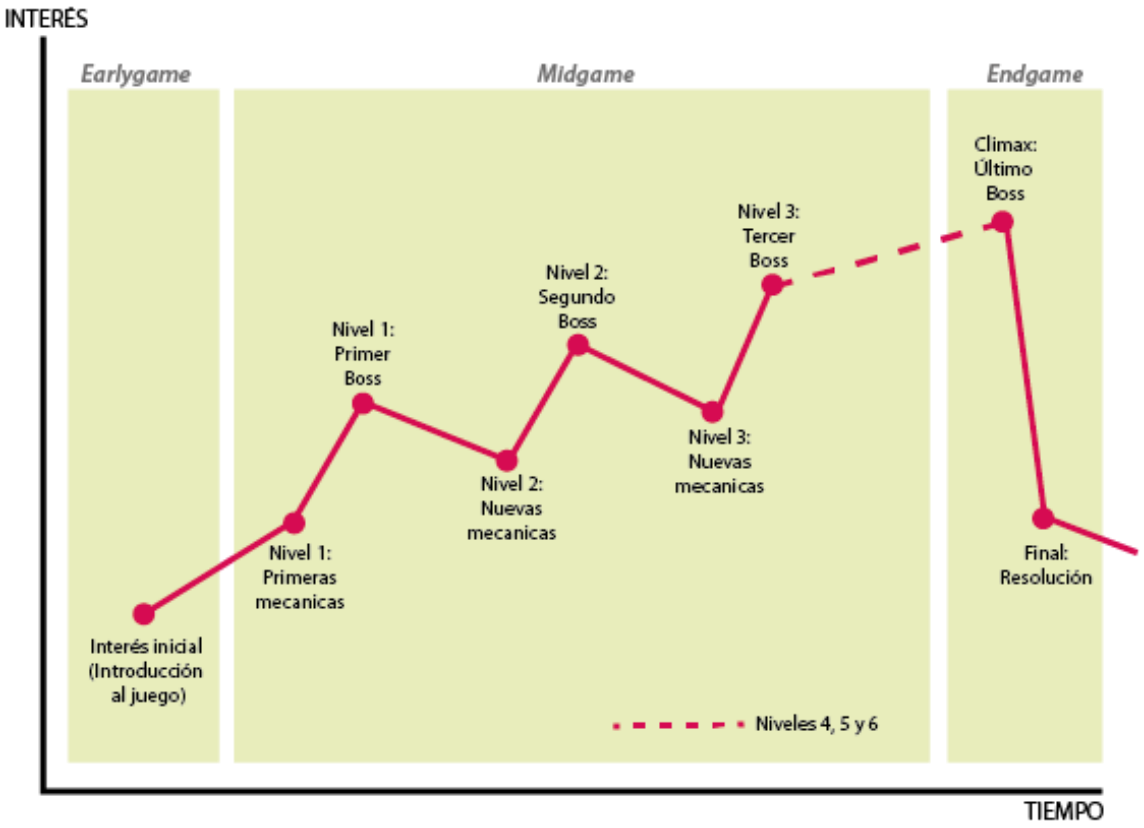

Ilustración Anexo 6 Curvas de interés

Earlygame:

Interés inicial, este inicio estaría vinculado al prólogo. Es muy importante crear un marco de referencia y, generar así, un lugar idóneo para que los jugadores puedan comenzar su aventura.

Midgame:

Primera toma de contacto real con el juego, las primeras mecánicas de este, primeras interacciones con el mundo y, por supuesto, los primeros enemigos. Se nos presentan los retos, misiones y recompensas. Una vez que hemos pasado este primer "subidón", pasamos un periodo de "relax transitorio", que nos prepara para las siguientes experiencias venideras.

Endgame:

Y en la última fase, en el fin del juego, es donde se decide todo. Vivimos un gran apogeo donde nos enfrentamos al reto final para conseguir la recompensa definitiva. Clímax, solución final y epílogo. El cierre de nuestro arco argumental, aquí tendremos el mayor punto de interés y donde se verán auténticamente reflejadas las intenciones de nuestro protagonista, así como la verdadera personalidad de nuestro Rey.

8. Personajes

8.1 Protagonista:

Pawn

Es un peón negro que quiere derrocar a su propio rey. Físicamente, es un peón clásico del tablero de ajedrez, pero caricaturizado y dotado de expresión facial gracias a unos grandes ojos, también tiene dos brazos, pero no tiene piernas. Puede moverse hacia delante y los lados dando pequeños saltitos, además puede saltar más alto para superar obstáculos.

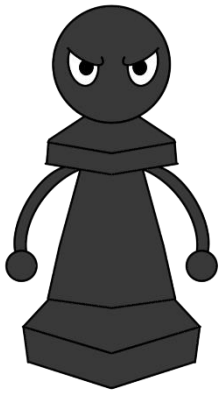

Ilustración Anexo 7 Diseño de Pawn

Sus ataques varían a lo largo del juego y son los siguientes:

Nivel 1: Pawn cuenta con una espada y al pulsar el clic izquierdo del ratón podrá atacar con ella. El ataque es un barrido horizontal.

Nivel 2: En este nivel se cambia de arma a una lanza y se añade un nuevo ataque: un clic del ratón será una estocada y dos un barrido. La segunda parte del nivel irá montado en un caballo, donde podrá moverse solamente horizontalmente y saltar.

Nivel 3: Pawn ahora tiene un escudo (clic derecho). Al usarlo, el personaje quedará completamente inmóvil.

Nivel 4: Sigue teniendo los ataques nombrados anteriormente, así como su escudo, la novedad en este nivel es que al saltar (espacio) se podrá volver a saltar en el aire, realizando un “doble salto”.

Nivel 5: En este nivel se puede usar un pequeño cañón que Pawn llevará bajo el brazo. Existe un *delay* entre disparos, este ataque es el que llega más lejos, pero también tiene más daño que los anteriores, aunque es más lento. Para apuntar se usa el clic derecho y para disparar el clic izquierdo. Cuando tenga equipado el cañón no podrá usar el escudo.

8.2 Antagonistas:

Peones:

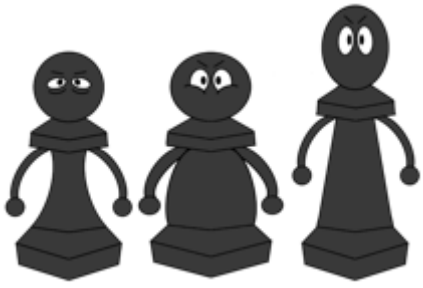

Ilustración Anexo 8 Diseño de los peones

Hay cuatro tipos de peones diferentes. Tres de ellos son peones básicos, la diferencia entre ellos y Pawn es que su físico: más delgado, más alto o rechoncho.

El ataque es igual que el ataque inicial de Pawn, pero atacarán de uno en uno dejando un margen de reacción al jugador. Cada uno de estos peones tendrá una característica diferente según su físico, siendo el delgado más rápido, el alto más fuerte y el rechoncho tiene más vida.

El cuarto tipo de peón es el jefe, que es mucho mayor en tamaño, fuerza y vida que los anteriores. Tiene pinchos que cubren su cabeza, cuello y base, además de unos guantes. Las características de sus ataques son los siguientes:

- **1ª fase (100% HP)**: Entra con una maza y sus ataques son poderosos y desde arriba.

- **2ª fase (50% HP)**: Se deshace de su espada y empieza a saltar y dar cabezazos contra el suelo para intentar aplastarte. Entre saltos habrá un tiempo en el que estará cansado y no podrá atacar de nuevo.

Ilustración Anexo 9
Diseño del peón jefe

Caballos:

Existen dos tipos de caballos diferentes en el juego: los estándares y el jefe. Lo componen el propio caballo y un peón jinete cuya arma es una lanza.

El caballo básico perseguirá a Pawn y saltará encima de él.

En cuanto al jefe, es mucho mayor en tamaño, fuerza y vida que los anteriores. Las características de sus ataques son los siguientes:

- **1ª fase (100% HP)**: Cuenta con una lanza, sus ataques son estocadas y saltos que irá intercambiando. Correrá hacia el personaje y se deberá conseguir que se golpee contra los pinchos de la pared para inmovilizarlo y atacarle.

- **2ª fase (50% HP)**: Lanza su arma, la cual comenzará a dar vueltas y perseguirá el jugador. Hay que seguir con lo mismo, hacer que se haga daño con los pinchos.

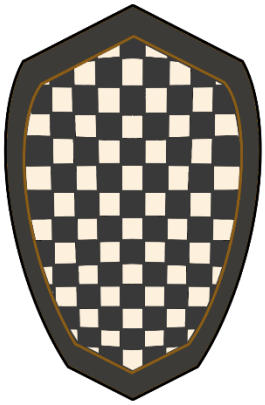

Ilustración Anexo 10
Diseño del escudo

Alfiles:

Existen 3 tipos de alfiles diferentes en el juego. Los alfiles son peones que han dedicado su vida al servicio de Dios, lo que comúnmente conocemos como Obispos. Estos tienen una mitra en la cabeza y su arma es un báculo.

Habrá un alfil verde y uno rojo. El primero curará a los enemigos mientras se protege del jugador; el rojo escapará de él al mismo tiempo que le lanza orbes de energía.

En cuanto al tercer alfil, este corresponde al jefe, el cual es mucho mayor en tamaño, fuerza y vida que los anteriores, las características de sus ataques son los siguientes:

Ilustración Anexo 11 Anexo 11. Diseño del alfil

- **Pre-Fase**: Aparecen alfiles y peones (devotos), y no sabes cuál es el alfil jefe. Una vez derrotados todos, se abren las puertas de la muralla y aparece realmente el Alfil Jefe.

- **1ª fase (100% HP)**: Su arma es un báculo, es rápido y además salta más alto que nosotros.

- **2ª fase (50% HP)**: Aparecen 4 alfiles que empiezan a curar poco a poco al alfil jefe y tendremos que acabar con ellos primero mientras evitamos los ataques del jefe (máximo 75% HP).

- **3ª fase (25% HP)**: Comienza también a hacer *dashes* para atacar, como los del Caballo Jefe, pero mucho más cortos y rápidos.

Torres:

Existen 2 tipos de torres diferentes en el juego. Están compuestas por una torre con un peón en su interior que dispara tres cañones. Son personajes inmóviles que se puede encontrar en el nivel y cuyos ataques hay que bloquear con el escudo. Habrá que subir las plataformas de alrededor suyo para llegar al botón de arriba y golpearle.

El último enemigo de este tipo es el jefe del nivel. Es mucho mayor en tamaño, fuerza y vida que los anteriores, y además puede moverse, las características de sus ataques son las siguientes:

- **1ª fase (100% HP)**: Su arma es un cañón, y tendremos que aprovechar el doble salto junto a los escombros que vayan cayendo para poder acercarnos a la cabeza y golpearla.

- **2ª fase (50% HP)**: Podrá realizar ráfagas de cañonazos que tendremos que evitar posicionándonos detrás de escombros.

- **3ª fase (25% HP)**: Balas explosivas que tienes que esquivarlas, en caso de cubrirte te harían mitad de daño y destruirá los escombros.

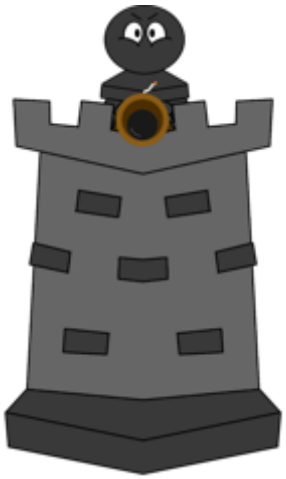

Ilustración Anexo 12 Diseño de la torre

Dama:

La dama es la figura más poderosa, solo hay una. Tiene una apariencia parecida a la de un peón, pero más esbelta y con una corona y un cetro mágico. Además, tiene mucha más vida que los demás.

- **1ª fase (100% HP)**: Sus armas son unas dagas que lanzará a distancia. Realizará un ataque giratorio si te acercas demasiado.

- **2ª fase (50% HP)**: Realiza una serie de ataques con orbes de energía mientras gira. Habrá que evitarlos para acercarse a ella y atacar cuando sea vulnerable.

- **3ª fase (25% HP)**: Se subirá un balcón y habrá que subir unas plataformas para llegar a ella y atacar. Mientras tanto, lanzará rayos en la dirección del jugador.

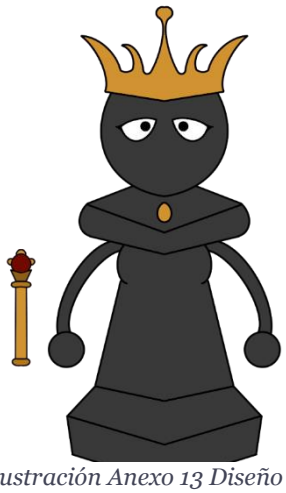

Ilustración Anexo 13 Diseño de la dama

Rey:

El rey lleva una larga capa roja y una corona de oro, es orgulloso y narcisista. Cree firmemente que la vida de los demás solo vale para proteger la suya, pero cuando no queda nadie para defenderlo, este huye, ya que en el fondo es un inútil y un cobarde, dejando atrás su corona y, por ende, renunciando a su reinado.

Ilustración Anexo 14 Diseño del rey

9. Jugabilidad

9.1 Descripción de los niveles:

Nivel 1	Presentación del juego y mecánicas de este. Habrá que enfrentarse a oleadas de enemigos y al primer jefe. Primera recompensa: Lanza.
Nivel 2	Cambio de jugabilidad, modo "carrera". Avanzamos por el escenario hasta llegar a un caballo, comenzando la parte en que se está montado a caballo y habrá que saltar y esquivar. Tras esta carrera aparecerá el segundo jefe. Segunda recompensa: Escudo.
Nivel 3	Vuelve las oleadas, esta vez con nuevos enemigos. Después seguirá el tercer jefe. Tercera recompensa: Doble salto.
Nivel 4	Último modo carrera, esta vez en sentido ascendente. Se terminará en la cima de una torre con el cuarto jefe. Cuarta recompensa: Cañón.
Nivel 5	Última fase de oleadas. Esta vez serán 3, después de cada una aparecerá un jefe. Quinta recompensa: cada jefe derrotado otorgará un fragmento de llave, al juntarlas se obtendrá la llave que abre las puertas del castillo.
Nivel 6	Clímax y resolución. Pelea contra el último jefe del juego. Tras vencerlo, dará paso al epílogo.

9.2 Metas y recompensas:

El objetivo principal de Pawn es avanzar por los diferentes niveles del juego hasta llegar al Rey, así que por el camino tendrá que ir eliminando a todo aquel que lo obstaculice. Estos enemigos no soltarán ningún objeto como recompensa, a excepción de los jefes de cada nivel. Por tanto, la vida que se pierda por el camino se recuperará solamente con los paquetes de vida que estarán repartidos de manera estratégica y solamente se podrá usar una vez, ya que cuando recuperes vida con uno de ellos este desaparecerá.

En cuanto a recompensas del juego, los jefes serán los encargados de proporcionarlos.

- **Peón**: Al derrotarlo te recompensará con una lanza.
- **Caballo**: Se obtendrá un escudo con el que se podrá protegerte de los ataques de los enemigos. Además, tu vida aumentará.
- **Alfil**: Conseguirás una mitra que dará la habilidad de doble salto, pudiendo acceder a zonas más altas que antes eran inaccesibles.
- **Torre**: Cañón que se podrá usar como arma secundaria para atacar a distancia.

9.3 Mecánicas y HUD:

Para la comodidad del jugador, existirán botones del teclado o mando para pelear o acceder de manera más rápida y sencilla a menús.

- **WASD / Joystick izquierdo**: Nos moveremos con las teclas “W” (delante), “A” (izquierda), “S” (detrás) y “D” (derecha); y con el eje del *joystick*.
- **Esc / botón pausa**: Se abrirá el menú de pausa, con un estilo parecido al de los recursos de debajo.

Ilustración Anexo 15 Ejemplo de diseño de menú

- **Espacio / X**: Si lo presionamos una vez pegaremos un salto simple, pero si ya hemos obtenido la habilidad de doble salto del alfil jefe podremos saltar más alto presionando este botón dos veces seguidas.
- **Clic izquierdo del ratón / R1**: Un solo clic realizará un barrido con la espada o lanza. En caso de tener la lanza, se añadirá otro ataque que podremos realizar dando doble clic, de esta manera podremos hacer una estocada contra algún enemigo. En caso de equiparse

el cañón, lanzaremos la munición (limitada) presionando una vez el clic izquierdo. Esta munición se te dará al principio de cada nivel.

- **Clic derecho del ratón / L1**: Una vez equipado el escudo, podremos usarlo presionando una vez esta tecla, pero solo se usará si lo mantenemos pulsado, una vez lo soltemos, el escudo volverá a posicionarse tras la espalda. En caso de tener equipado el cañón, este botón nos servirá para apuntar.

- **E / R2**: Cambio de arma. Podremos cambiar de lanza y escudo a cañón, o viceversa. Los ejemplos que podemos ver a continuación estarían situados en la parte inferior derecha de la pantalla

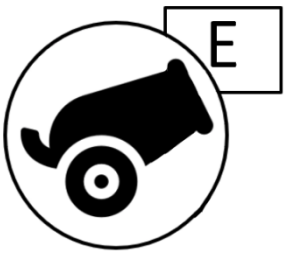

Ilustración Anexo 16 Imagen del cañón

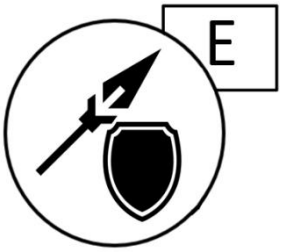

Ilustración Anexo 17 Imagen del escudo y lanza

- **Desplazar el ratón / Joystick derecho**: La cámara seguirá en todo momento al personaje y se moverá a la par con el ratón o *joystick*.

La **vida** está situada abajo a la izquierda y representada por la silueta de Pawn, que irá desapareciendo conforme recibimos daño, parecido a la barra de vida usada en *Minecraft*.

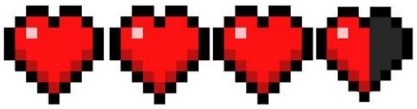

Ilustración Anexo 18 Ejemplo de vida

En cuanto a la de los **jefes finales**, se encontrará en la parte superior central de la pantalla, representado con el color rojo y encima, el nombre del enemigo, como en este ejemplo de *Borderlands 3*.

Ilustración Anexo 19 Captura del videojuego Bordelands 3

Al **apuntar** con el cañón a un enemigo la mira se volverá de color rojo cuando el enemigo pueda ser herido con el disparo, si sale en blanco significa que no está dentro del alcance o que no hay ningún enemigo en la mira. Todo esto se hará en tercera persona. La mira seguirá al ratón. Se pueden tomar como ejemplo las imágenes siguientes del juego *Worms 3D*, pero cambiando la primera persona por una cámara en tercera.

Ilustración Anexo 20 Captura 1 del juego Worms 3D

Ilustración Anexo 21 Captura 2 del juego Worms 3D

10. Diagrama de estados:

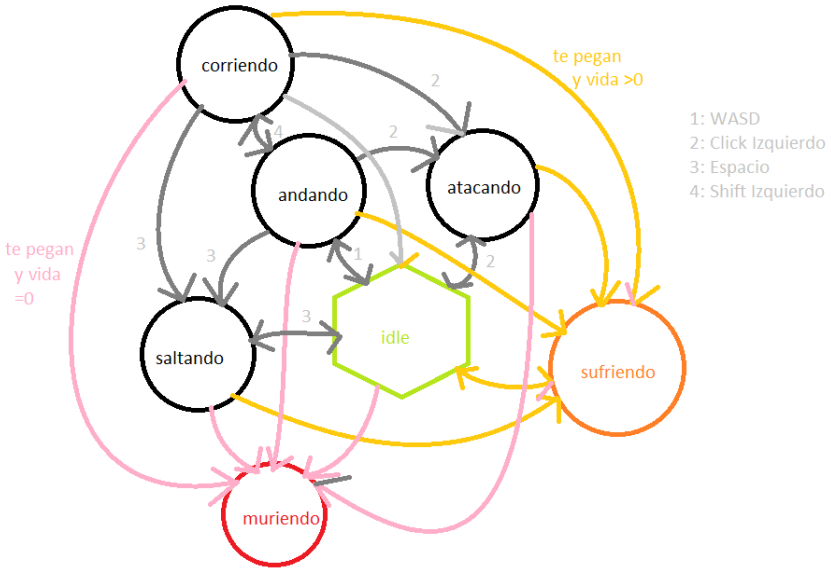

Ilustración Anexo 22 Máquina de estados

11. Reglas del juego:

- El jugador tendrá que jugar dentro del escenario, no podrá rebasar los límites de este, aunque lo intente.
- Si muere, empezará desde el último punto de guardado.
- No se puede mover ni usar la lanza o cañón al mismo tiempo que el escudo.
- Una vez se utilice un paquete de vida, este desaparecerá y no se podrá volver a usar.
- No se podrá volver a los niveles anteriores.

12. Menús y navegabilidad

12. 1 Diagrama de navegación

El juego contará con 2 menús: el inicial y el de pausa. La navegabilidad entre cada uno de ellos la podemos ver de manera esquematizada en la siguiente imagen.

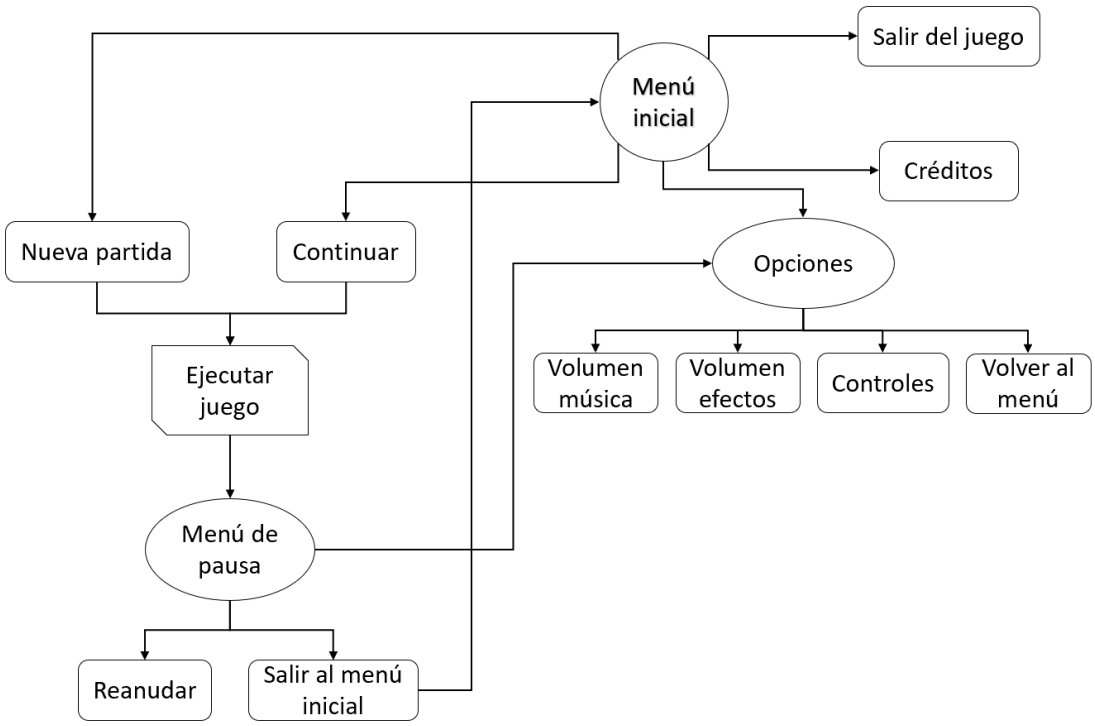

Ilustración Anexo 23 Navegación entre menús

12.2 Menú inicial:

Es el que veremos nada más abramos el juego. Una vez dentro, se observará la animación de un libro abriéndose, para a continuación mostrar las diferentes opciones del menú:

- **Nueva partida**: Te mostrará un mensaje preguntando si deseas comenzar partida o no. En caso de que la respuesta sea afirmativa, te llevará directamente al inicio del juego, específicamente en la narrativa de la historia.

- **Cargar partida**: Continuaremos por donde nos quedamos en el juego (el último punto de guardado).
- **Multijugador**: Se abrirá el *lobby* del este modo de juego
- **Opciones**: Se abrirá un menú para poder configurar varias opciones.
- **Créditos**: Se mostrará por pantalla los créditos del juego.
- **Salir del juego**: Se saldrá del menú.

12.3 Menú de pausa:

Se accederá a él presionando la tecla esc. Una vez abierto, el fondo del juego se verá difuminado y un poco más oscuro, de esta manera se mostrará varias opciones:

- **Reanudar**: Se cerrará el menú y podremos seguir jugando.
- **Cargar partida**: Cargará una partida guardada.
- **Opciones**: Compartirá el mismo menú de opciones del menú inicial.
- **Menú principal**: Nos llevará al menú inicial.

12. 4 Menú opciones:

Esta ventana será la que nos permitirá configurar algunas cosas del juego si es que no estamos a gusto con la configuración que viene por defecto. Por ello, contará con las siguientes opciones:

- **Gráficos**
- **Sonido**
    - Volumen global
    - Volumen de la música
    - Volumen de los efectos de sonido (SFX)
    - Volumen del ambiente del juego
- **Controles**
- **Volver:** Se vuelve al menú anterior

13. Escenarios

13.1 Estética de la ambientación

El juego se desarrolla en una guerra medieval en un mundo de juegos de mesa. Existen dos entornos dentro de este videojuego, el primero es el mundo jugable y nuestro contexto más cercano, y en el segundo veremos el mundo externo a nuestra guerra, donde podremos observar representaciones de otros juegos de mesa.

Como hemos comentado anteriormente, la estética visual será una guerra medieval donde los colores predominantes son el marrón y el verde, pero también podremos observar otros colores

algo más vibrantes cuyo papel es darle vida a los escenarios, puesto que el color predominante de los personajes es el negro y queremos generar cierto contraste entre ellos.

Tanto escenarios como personajes serán low poly, ya que nuestro objetivo es lograr una estética cartoon y minimalista. Utilizaremos colores similares a los de la siguiente paleta.

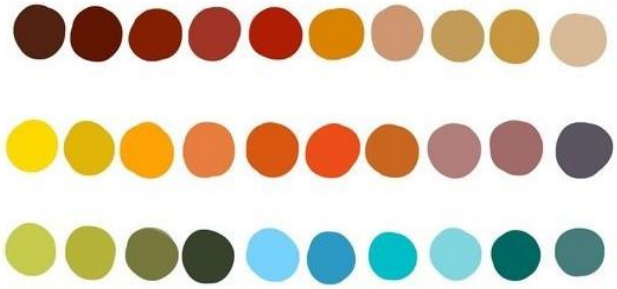

Ilustración Anexo 24 Paleta de colores

13.2 Escenarios existentes por niveles y escenas

Existen 4 escenarios donde se desarrolla el videojuego. A su vez, en estos pueden desarrollarse varios niveles y habrá diferentes escenas dentro de cada escenario. A continuación, detallaremos qué engloba a cada uno de ellos:

- **Escenario 1**, **el campo de batalla**: engloba los niveles 1, 2 y 3 del juego. En este escenario habrá 7 escenas diferentes, una escena inicial de cada nivel donde tendremos que avanzar y derrotar a los enemigos, y otra escena para cada uno de los jefes finales. En el nivel 2 se divide en tres escenas.

- **Escenario 2, la torre**: corresponde al nivel 4 del juego. En este escenario habrá 2 escenas, una para el nivel y otra para el *boss*.

- **Escenario 3, el patio de armas**: corresponde al nivel 5 del juego. En este escenario habrá 2 escenas, una para el nivel y otra para el jefe.

- **Escenario 4, interior del castillo**: corresponde al nivel 6 del juego. En este último escenario habrá 3 escenas, la primera será una presentación de la Dama y el Rey, la segunda será la batalla y derrota de la Dama, y la última será el final del juego.

13.3 Breve descripción de los escenarios

- **Escenario 1**, **el campo de batalla**: este escenario tiene diferentes escenas que representan cada nivel, la primera será el campo de batalla, en una línea casi recta donde habrá enemigos, además de diferentes obstáculos como pueden ser: árboles, cajas, trincheras, barricadas, puestos de arma, vallas, etc. que también serán los que delimitarán el escenario.

  La segunda escena será un camino en línea recta, donde iremos a caballo y tendremos que saltar obstáculos y evitar o golpear enemigos. Estos obstáculos serán los mismos que en la escena anterior y tendrán la misma función. El jefe final estará en un área circular delimitada.

  La tercera escena estará a las afueras del muro del castillo y será un área rectangular, al igual que las escenas anteriores contaremos con obstáculos y enemigos, que limitan o cortan el paso de nuestro protagonista.

- **Escenario 2, la torre:** este escenario será circular y tendremos que ascender por las escaleras de la torre que suben en espiral hasta el matacán donde nos encontraremos al

jefe de este nivel. El escenario estará compuesto por enemigos y obstáculos como cajas de armas, huecos entre escalones que tendremos que saltar, etc. En la escena del matacán encontraremos escombros, cajas, muros, etc.

- **Escenario 3, el patio de armas**: será un área circular que abarca desde las puertas, murallas y torres defensivas hasta la entrada al castillo. Los obstáculos serán escasos, ya que es un nivel en el que estaremos luchando constantemente.

- **Escenario 4, interior del castillo**: tendrá forma rectangular y representa la sala del trono, en él podremos apreciar objetos que servirán para delimitar el escenario, así como para protegernos y ambientar la escena, puede ser: bancos, muros y columnas de piedra, etc.

14. Elementos jugables

Pawn es un videojuego cuyos elementos jugables son limitados, estos podrían resumirse en:

- **Los enemigos**: cuya interacción se resume en ser golpeados o golpearnos hasta que son derrotados.

- **Paquetes de vida**: los cuales estarán repartidos estratégicamente por los escenarios y al pasar por encima nos recuperarán automáticamente un porcentaje de vida. Una vez utilizados estos desaparecen, por lo que son de un solo uso, además, si nuestra vida está al 100% no podremos curarnos más.

- **Obstáculos**: estos no generan una interacción automática como los demás, pero podemos saltar encima y acceder a una zona más alejada, o para cubrirnos del fuego enemigo, para delimitar el terreno o simplemente para decorar el escenario. Los *prefabs* de los que estamos hablando son los siguientes: Árboles (existen 3 tipos), Carros (existen 2 tipos), hogueras, tiendas de campaña, camas y cojines, barriles (dos tipos), rocas (al menos 5 tipos diferentes), carteles orientativos, troncos caídos y tocones, edificios (5 diferentes), un pozo, vallas, hierba, caballos, huerta y un puente que cruza el río.

15. Bocetos

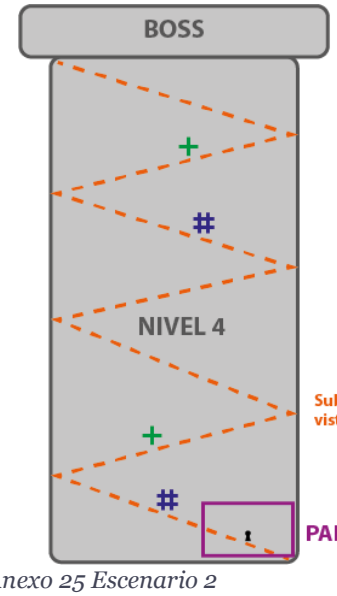

Ilustración Anexo 25 Escenario 2

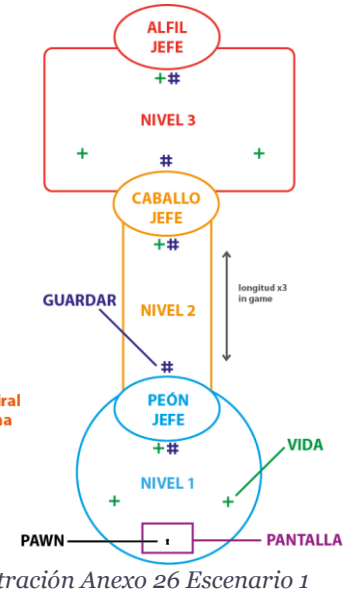

Ilustración Anexo 26 Escenario 1

Ilustración Anexo 27 Escenario 4

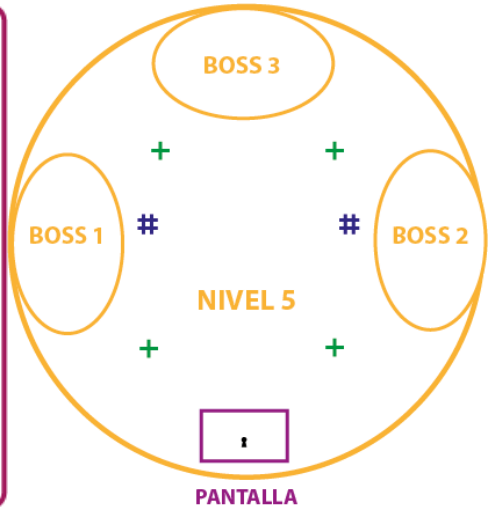

Ilustración Anexo 28 Escenario 3

Anexo II: Árboles de comportamiento de todos los enemigos

1. Peón básico / Caballo básico

Utilizan el mismo, a excepción del BTid.

El alfil rojo tendrá el mismo, pero sin el subárbol “ChasePlayerT”.

El Caballo Jefe también tendrá el mismo, pero en este caso, sin el subárbol “PatrollingT”.

```
<?xml version="1.0" encoding="utf-8" ?>

<BT>
    <BTid>EnemyAIBT</BTid>
    <Bt>
        <Trees>
            <Tree Root="YES">
                <Name>Root</Name>
                <Child_Nodes>
                    <CN>
                        <CN_Type>node</CN_Type>
                        <CN_Name>sequence</CN_Name>
                        <Child_Nodes>
                            <CN>
                                <CN_Type>tree</CN_Type>
                                <CN_Name>CheckingT</CN_Name>
                            </CN>
                            <CN>
                                <CN_Type>node</CN_Type>
                                <CN_Name>fallback</CN_Name>
                                <Child_Nodes>
                                    <CN>
                                        <CN_Type>tree</CN_Type>
                                        <CN_Name>PatrollingT</CN_Name>
                                    </CN>
                                    <CN>
                                        <CN_Type>tree</CN_Type>
                                        <CN_Name>ChasePlayerT</CN_Name>
                                    </CN>
                                    <CN>
                                        <CN_Type>tree</CN_Type>
                                        <CN_Name>AttackPlayerT</CN_Name>
                                    </CN>
                                </Child_Nodes>
                            </CN>
                        </Child_Nodes>
                    </CN>
                </Child_Nodes>
            </Tree>
```

Ilustración Anexo 29 XML del BT del Peón Parte 1

```
<Tree Root="NO">
    <Name>CheckingT</Name>
    <Child_Nodes>
        <CN>
            <CN_Type>node</CN_Type>
            <CN_Name>sequence</CN_Name>
            <Child_Nodes>
                <CN>
                    <CN_Type>node</CN_Type>
                    <CN_Name>Checking</CN_Name>
                </CN>
            </Child_Nodes>
        </CN>
    </Child_Nodes>
</Tree>

<Tree Root="NO">
    <Name>PatrollingT</Name>
    <Child_Nodes>
        <CN>
            <CN_Type>node</CN_Type>
            <CN_Name>while _toPatrol</CN_Name>
            <Child_Nodes>
                <CN>
                    <CN_Type>node</CN_Type>
                    <CN_Name>sequence</CN_Name>
                    <Child_Nodes>
                        <CN>
                            <CN_Type>node</CN_Type>
                            <CN_Name>Patrolling</CN_Name>
                        </CN>
                    </Child_Nodes>
                </CN>
            </Child_Nodes>
        </CN>
    </Child_Nodes>
</Tree>
```

Ilustración Anexo 30 XML del BT del Peón Parte 2

```
            <Tree Root="NO">
                <Name>ChasePlayerT</Name>
                <Child_Nodes>
                    <CN>
                        <CN_Type>node</CN_Type>
                        <CN_Name>while _toChase</CN_Name>
                        <Child_Nodes>
                            <CN>
                                <CN_Type>node</CN_Type>
                                <CN_Name>sequence</CN_Name>
                                <Child_Nodes>
                                    <CN>
                                        <CN_Type>node</CN_Type>
                                        <CN_Name>ChasePlayer</CN_Name>
                                    </CN>
                                </Child_Nodes>
                            </CN>
                        </Child_Nodes>
                    </CN>
                </Child_Nodes>
            </Tree>
            <Tree Root="NO">
                <Name>AttackPlayerT</Name>
                <Child_Nodes>
                    <CN>
                        <CN_Type>node</CN_Type>
                        <CN_Name>while _toAttack</CN_Name>
                        <Child_Nodes>
                            <CN>
                                <CN_Type>node</CN_Type>
                                <CN_Name>sequence</CN_Name>
                                <Child_Nodes>
                                    <CN>
                                        <CN_Type>node</CN_Type>
                                        <CN_Name>AttackPlayer</CN_Name>
                                    </CN>
                                </Child_Nodes>
                            </CN>
                        </Child_Nodes>
                    </CN>
                </Child_Nodes>
            </Tree>

        </Trees>
    </Bt>
</BT>
```

Ilustración Anexo 31 XML del BT del Peón Parte 3

2. Alfil verde

```
<?xml version="1.0" encoding="utf-8" ?>

<BT>
    <BTid>EnemyAlfilVerdeAIBT</BTid>
    <Bt>
        <Trees>
      <Tree Root="YES">
                <Name>Root</Name>
        <Child_Nodes>
                    <CN>
                        <CN_Type>node</CN_Type>
            <CN_Name>sequence</CN_Name>
            <Child_Nodes>
                            <CN>
                <CN_Type>tree</CN_Type>
                <CN_Name>CheckingT</CN_Name>
                            </CN>
                <CN>
                    <CN_Type>node</CN_Type>
                    <CN_Name>fallback</CN_Name>
                    <Child_Nodes>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>SupportingT</CN_Name>
                        </CN>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>PatrollingT</CN_Name>
                        </CN>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>HidingT</CN_Name>
                        </CN>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>LookingT</CN_Name>
                        </CN>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>AttackPlayerT</CN_Name>
                        </CN>
                    </Child_Nodes>
                </CN>
            </Child_Nodes>
                    </CN>
      </Child_Nodes>
          </Tree>
```

Ilustración Anexo 32 XML del BT del Alfil Verde Parte 1

```
    <Tree Root="NO">
      <Name>CheckingT</Name>
      <Child_Nodes>
        <CN>
          <CN_Type>node</CN_Type>
          <CN_Name>sequence</CN_Name>
          <Child_Nodes>
            <CN>
              <CN_Type>node</CN_Type>
              <CN_Name>Checking</CN_Name>
            </CN>
          </Child_Nodes>
        </CN>
      </Child_Nodes>
    </Tree>

  <Tree Root="NO">
      <Name>SupportingT</Name>
      <Child_Nodes>
          <CN>
              <CN_Type>node</CN_Type>
              <CN_Name>while _toSupport</CN_Name>
              <Child_Nodes>
                <CN>
                  <CN_Type>node</CN_Type>
                  <CN_Name>sequence</CN_Name>
                  <Child_Nodes>
                    <CN>
                      <CN_Type>node</CN_Type>
                      <CN_Name>Supporting</CN_Name>
                    </CN>
                  </Child_Nodes>
                </CN>
              </Child_Nodes>
          </CN>
      </Child_Nodes>
    </Tree>
```

Ilustración Anexo 33 XML del BT del Alfil Verde Parte 2

```
    <Tree Root="NO">
      <Name>PatrollingT</Name>
      <Child_Nodes>
          <CN>
              <CN_Type>node</CN_Type>
              <CN_Name>while _toPatrol</CN_Name>
              <Child_Nodes>
                <CN>
                  <CN_Type>node</CN_Type>
                  <CN_Name>sequence</CN_Name>
                  <Child_Nodes>
                    <CN>
                      <CN_Type>node</CN_Type>
                      <CN_Name>Patrolling</CN_Name>
                    </CN>
                  </Child_Nodes>
                </CN>
              </Child_Nodes>
          </CN>
      </Child_Nodes>
    </Tree>

  <Tree Root="NO">
      <Name>HidingT</Name>
      <Child_Nodes>
          <CN>
              <CN_Type>node</CN_Type>
              <CN_Name>while _toHide</CN_Name>
              <Child_Nodes>
                <CN>
                  <CN_Type>node</CN_Type>
                  <CN_Name>sequence</CN_Name>
                  <Child_Nodes>
                    <CN>
                      <CN_Type>node</CN_Type>
                      <CN_Name>HideFromPlayer</CN_Name>
                    </CN>
                  </Child_Nodes>
                </CN>
              </Child_Nodes>
          </CN>
      </Child_Nodes>
    </Tree>
```

Ilustración Anexo 34 XML del BT del Alfil Verde Parte 3

```
<Tree Root="NO">
    <Name>LookingT</Name>
    <Child_Nodes>
        <CN>
            <CN_Type>node</CN_Type>
            <CN_Name>while _toLook</CN_Name>
            <Child_Nodes>
              <CN>
                <CN_Type>node</CN_Type>
                <CN_Name>sequence</CN_Name>
                <Child_Nodes>
                  <CN>
                    <CN_Type>node</CN_Type>
                    <CN_Name>LookingAtPlayer</CN_Name>
                  </CN>
                </Child_Nodes>
              </CN>
            </Child_Nodes>
        </CN>
    </Child_Nodes>
  </Tree>
 <Tree Root="NO">
    <Name>AttackPlayerT</Name>
    <Child_Nodes>
        <CN>
            <CN_Type>node</CN_Type>
            <CN_Name>while _toAttack</CN_Name>
            <Child_Nodes>
              <CN>
                <CN_Type>node</CN_Type>
                <CN_Name>sequence</CN_Name>
                <Child_Nodes>
                  <CN>
                    <CN_Type>node</CN_Type>
                    <CN_Name>AttackPlayer</CN_Name>
                  </CN>
                </Child_Nodes>
              </CN>
            </Child_Nodes>
        </CN>
    </Child_Nodes>
  </Tree>
    </Trees>
</Bt>
</BT>
```

Ilustración Anexo 35 XML del BT del Alfil Verde Parte 4

3. Reina

```
<?xml version="1.0" encoding="utf-8" ?>

<BT>
    <BTid>EnemyReinaAIBT</BTid>
    <Bt>
        <Trees>
      <Tree Root="YES">
                <Name>Root</Name>
        <Child_Nodes>
                    <CN>
                        <CN_Type>node</CN_Type>
            <CN_Name>sequence</CN_Name>
            <Child_Nodes>
                            <CN>
                <CN_Type>tree</CN_Type>
                <CN_Name>CheckingT</CN_Name>
                            </CN>
                <CN>
                    <CN_Type>node</CN_Type>
                    <CN_Name>fallback</CN_Name>
                    <Child_Nodes>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>ChasePlayerT</CN_Name>
                        </CN>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>AttackPlayerT</CN_Name>
                        </CN>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>SpinningT</CN_Name>
                        </CN>
                        <CN>
                            <CN_Type>tree</CN_Type>
                            <CN_Name>LightningT</CN_Name>
                        </CN>
                    </Child_Nodes>
                </CN>
            </Child_Nodes>
                    </CN>
        </Child_Nodes>
            </Tree>
```

Ilustración Anexo 36 XML del BT de la Reina Parte 1

```
    <Tree Root="NO">
      <Name>CheckingT</Name>
      <Child_Nodes>
        <CN>
          <CN_Type>node</CN_Type>
          <CN_Name>sequence</CN_Name>
          <Child_Nodes>
            <CN>
              <CN_Type>node</CN_Type>
              <CN_Name>Checking</CN_Name>
            </CN>
          </Child_Nodes>
        </CN>
      </Child_Nodes>
    </Tree>

  <Tree Root="NO">
      <Name>ChasePlayerT</Name>
      <Child_Nodes>
          <CN>
              <CN_Type>node</CN_Type>
              <CN_Name>while _toChase</CN_Name>
              <Child_Nodes>
                <CN>
                  <CN_Type>node</CN_Type>
                  <CN_Name>sequence</CN_Name>
                  <Child_Nodes>
                    <CN>
                      <CN_Type>node</CN_Type>
                      <CN_Name>ChasePlayer</CN_Name>
                    </CN>
                  </Child_Nodes>
                </CN>
              </Child_Nodes>
          </CN>
      </Child_Nodes>
    </Tree>
```

Ilustración Anexo 37 XML del BT de la Reina Parte 2

```
    <Tree Root="NO">
      <Name>AttackPlayerT</Name>
      <Child_Nodes>
          <CN>
              <CN_Type>node</CN_Type>
              <CN_Name>while _toAttack</CN_Name>
              <Child_Nodes>
                <CN>
                  <CN_Type>node</CN_Type>
                  <CN_Name>sequence</CN_Name>
                  <Child_Nodes>
                    <CN>
                      <CN_Type>node</CN_Type>
                      <CN_Name>AttackPlayer</CN_Name>
                    </CN>
                  </Child_Nodes>
                </CN>
              </Child_Nodes>
          </CN>
      </Child_Nodes>
    </Tree>

  <Tree Root="NO">
      <Name>SpinningT</Name>
      <Child_Nodes>
          <CN>
              <CN_Type>node</CN_Type>
              <CN_Name>while _toSpin</CN_Name>
              <Child_Nodes>
                <CN>
                  <CN_Type>node</CN_Type>
                  <CN_Name>sequence</CN_Name>
                  <Child_Nodes>
                    <CN>
                      <CN_Type>node</CN_Type>
                      <CN_Name>Spinning</CN_Name>
                    </CN>
                  </Child_Nodes>
                </CN>
              </Child_Nodes>
          </CN>
      </Child_Nodes>
    </Tree>
```

Ilustración Anexo 38 XML del BT de la Reina Parte 3

```
    <Tree Root="NO">
        <Name>LightningT</Name>
        <Child_Nodes>
            <CN>
                <CN_Type>node</CN_Type>
                <CN_Name>while _toLightning</CN_Name>
                <Child_Nodes>
                  <CN>
                    <CN_Type>node</CN_Type>
                    <CN_Name>sequence</CN_Name>
                    <Child_Nodes>
                      <CN>
                        <CN_Type>node</CN_Type>
                        <CN_Name>DisparарRayos</CN_Name>
                      </CN>
                    </Child_Nodes>
                  </CN>
                </Child_Nodes>
            </CN>
        </Child_Nodes>
      </Tree>
        </Trees>
    </Bt>
</BT>
```

Ilustración Anexo 39 XML del BT de la Reina Parte 4

Anexo III

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**	**X**			
ODS 2. **Hambre cero.**		**X**		
ODS 3. **Salud y bienestar.**	**X**			
ODS 4. **Educación de calidad.**	**X**			
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**		**X**		
ODS 7. **Energía asequible y no contaminante.**			**X**	
ODS 8. **Trabajo decente y crecimiento económico.**			**X**	
ODS 9. **Industria, innovación e infraestructuras.**				**X**
ODS 10. **Reducción de las desigualdades**.	**X**			
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**		**X**		
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**		**X**		
ODS 16. **Paz, justicia e instituciones sólidas.**	**X**			
ODS 17. **Alianzas para lograr objetivos.**	**X**			

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

De los objetivos de desarrollo sostenibles que aparecen en la tabla de la página anterior, en este TFG destacan:

- **Educación de calidad**. Pues ofrece información útil sobre cómo usar la herramienta GAIA para el posible desarrollo de la inteligencia artificial de un videojuego en Unity, y lo hace a través de un ejemplo práctico como es Pawn. También aporta conceptos básicos de animación, tanto de Blender como del propio motor de Unity.

  En los siguientes ODS mencionados se podrá observar como el juego también trata de transmitir un mensaje educativo con relación a la resolución de conflictos, a la búsqueda de la igualdad de derechos y a las consecuencias principales de la guerra.
- **Paz, justicia e instituciones sólidas**. La temática principal de Pawn es finalizar una guerra que está acabando con la vida de los ciudadanos de ambos bandos. Para ello, el jugador tendrá que derrotar a su propio rey, uno de los causantes del conflicto. Es por ello por lo que el núcleo central del apartado narrativo del juego se basa en hacer justicia, con el fin de conseguir la paz entre las dos naciones implicadas.

  Siguiendo el hilo de este ODS, aparecen otros que, si bien no tienen una relevancia tan presente en el TFG, forman parte de las consecuencias directas de la guerra y también motivan al personaje principal a llevar a cabo su plan:
  - Salud y bienestar / Fin de la pobreza / Hambre cero / Trabajo decente y crecimiento económico. En tiempos de guerra, al menos dentro del contexto medieval donde se sitúa el juego, se suelen utilizar gran parte de los recursos de la nación como suministros para el ejército. Esto puede provocar una privación de bienes esenciales hacia el resto de los ciudadanos, como puede ser la comida o la vivienda.
  - Vida de ecosistemas terrestres / Agua limpia y saneamiento / Acción por el clima / Energía asequible y no contaminante. Por otro lado, la guerra también afecta al medioambiente y a su deterioro. Hoy más que nunca, en un contexto de emergencia climática donde el uso de armas bioquímicas o nucleares puede resultar en consecuencias desastrosas para el planeta, estos ODS toman un papel especialmente importante en cualquier sociedad que busca preservar su naturaleza, tanto por su conservación como por la obtención de recursos para la supervivencia humana.
  - Reducción de las desigualdades. El personaje de Pawn es un peón, pieza que forma parte de la primera línea de batalla y suele ser de las primeras en morir. Es por este motivo que Pawn decide escalar por el sistema de jerarquías compuesto por las piezas del ajedrez hasta llegar al rey. La intención de este plan no es solo

acabar con la guerra, sino también demostrar que hasta la más significante de las piezas es capaz de acabar con un sistema que oprime a su pueblo. Así pues, el derrotar al rey no es solamente una solución a la guerra, sino un símbolo de libertad.

- **Alianzas para lograr objetivos**. Este TFG se ha realizado de manera conjunta con Jennifer Canuto Soler y Sergio Muñoz Alejandre, motivo por el que creo que este proyecto promueve este ODS y la colaboración entre alumnos para un trabajo coordinado.