UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Black Friday War: Implementación de las mecánicas de los enemigos de un videojuego

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Fuentes García, Diego Manuel

Tutor/a: Abad Cerdá, Francisco José

CURSO ACADÉMICO: 2022/2023

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Black Friday War: Implementación de las mecánicas de los enemigos de un videojuego

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Diego Manuel Fuentes García

Tutor: Francisco José Abad Cerdá

2022-23

Resumen

Black Friday War es un videojuego 2D de vista cenital enmarcado en el género roguelike. El jugador controlará a un grupo de cuatro personajes a los que guiará a través de tres niveles. En cada nivel, el jugador se enfrentará a una serie de enemigos a medida que busca tres objetos clave que le permitirán luchar contra el enemigo final del nivel.

Este Trabajo Fin de Grado consistirá en el desarrollo de las mecánicas de los distintos tipos de enemigos, usando para ello técnicas de Inteligencia Artificial, entre las que se explorarán las State Machines y los Behaviour Trees. También se estudiará la posibilidad de desarrollar acciones coordinadas o tácticas por parte de los grupos de enemigos.

Palabras clave: Videojuegos; 2D; roguelike; enemigos; Inteligencia Artificial.

Abstract

Black Friday War is a 2D top-down video game framed in the roguelike genre. The player will control a group of four characters whom he will guide through three levels. In each level, the player will face a series of enemies as they search for three key items that will allow them to fight the final enemy of the level.

This Final Degree Project will consist of the development of the mechanics of the different types of enemies, using Artificial Intelligence techniques, among which State Machines and Behavior Trees will be explored. The possibility of developing coordinated or tactical actions by enemy groups will also be studied.

Keywords: Video games, 2D, roguelike, enemies, Artificial Intelligence.

Agradecimientos

A mis padres por su apoyo durante la carrera.

A mis compañeros Francisco José Noguera Guijarro y Jorge Duart Marzo, sin los que este videojuego no hubiera llegado a hacerse realidad.

A mi tutor Paco Abad por sus consejos y su apoyo a lo largo de este Proyecto.

Gracias a todos.

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

El proyecto desarrollado en este Trabajo de Tin de Grado es un videojuego. Los videojuegos son aplicaciones dirigidas normalmente a entretener a los usuarios [1]. Los primeros videojuegos aparecieron durante los años 1950s, entre los que se destacan OXO (1952), una especie de versión computarizada del tres en raya, o Tennis for two (1958), un simulador de tenis de mesa, aunque por aquel entonces los videojuegos se encontraban relegados en los entornos de investigación. En el año 1966 se comenzó a desarrollar el proyecto Fox and Hounds que daría inicio al videojuego doméstico. Finalmente, en la década de 1970s se inició la ascensión de los videojuegos, apareciendo los salones recreativos y las primeras consolas, iniciándose así la expansión y evolución de este sector, que aún sigue presente en nuestra época y cuyo crecimiento parece no terminar [2].

Nuestro videojuego, Black Friday War, se enmarca en el género Roguelike, el cual se caracteriza por la exploración de niveles laberínticos, denominados mazmorras, que se generan aleatoriamente en cada partida.

1.1. Motivación

En los últimos años, el protagonismo de los videojuegos en nuestra sociedad no ha hecho más que aumentar, algo que pudo observarse durante la crisis del COVID-19, en la que se produjo un crecimiento del consumo de este tipo de productos, como recoge IDG Consulting en su informe "Análisis del impacto del impacto del COVID-19 y mejores prácticas en el sector de los videojuegos" [3].

Entre los muchos géneros de los videojuegos, uno de los más interesantes y cuya popularidad está en auge es el género Roguelike. No conocí dicho género hasta hace unos años, cuando adquirí el título Enter The Gungeon, un frenético Roguelike en el que el jugador encarna a un pistolero que explora una mazmorra llena de monstruos y trampas. Rápidamente la propuesta consiguió encandilarme y empecé a interesarme más por este tipo de títulos.

El campo de la IA tiene una relación bastante estrecha con los videojuegos, sobre todo en lo que respecta a los personajes no jugadores (NPC) de un videojuego. Bien sea porque estos personajes ayudan al jugador en el cumplimiento de sus objetivos o justo lo contrario, tomando el papel de los rivales y enemigos del jugador, los NPC son una parte fundamental de la gran mayoría de los videojuegos y, por lo tanto, su implementación conlleva una gran relevancia en el desarrollo de videojuegos.

1.2. Objetivos

Los objetivos de este trabajo de fin de grado son los siguientes:

- Desarrollar un proyecto software haciendo uso de metodologías ágiles.
- Aprender a utilizar técnicas de Inteligencia Artificial.
- Implementar los enemigos de un videojuego, teniendo en cuenta la posibilidad de construir una IA táctica que coordine a los enemigos.

1.3. Metodología

Para llevar a cabo este proyecto se ha hecho uso de metodologías ágiles, basándonos principalmente en la metodología Scrum. La principal razón de escoger este tipo de metodologías es la flexibilidad que ofrecen frente a las metodologías tradicionales, permitiendo modificar la

planificación del proyecto durante su desarrollo. Esto puede ser de gran utilidad en entornos de gran incertidumbre, como pueden ser en ocasiones el desarrollo de videojuegos. No es ninguna novedad la existencia de desarrollos caóticos en el sector, llegando a retrasarse la salida de algunos títulos e incluso a cancelarse proyectos.

Otra de las razones de la elección de dicho tipo de metodologías, es la existencia de cierta experiencia en estas metodologías gracias al desarrollo de proyectos en asignaturas de la rama de ingeniería del software.

1.4. Estructura de la memoria

Esta estructura se vertebra en siete apartados, siendo el primero la introducción, en la que se presenta el proyecto de este TFG. Después encontramos el estado del arte, en el que se enmarca el videojuego dentro de su género y se investigan las distintas técnicas de IA utilizadas en el sector. El tercer apartado corresponde al análisis del problema, en la que se estudian las tareas o requisitos a cumplir para realizar el proyecto. El cuarto apartado es el del diseño, que describe la estructura de la aplicación, y el quinto es el de desarrollo, centrado en la realización del proyecto. Por último, encontramos los apartados de implementación, que analiza el resultado obtenido tras el desarrollo, y las conclusiones, que incluyen los posibles trabajos futuros a partir del proyecto realizado.

Al final de la memoria se encuentra la bibliografía con las referencias a las obras consultadas para realizar este y un glosario con las definiciones de las abreviaturas utilizadas. También podemos encontrar los apéndices con información adicional del proyecto, como el GDD del videojuego Black Friday War y el documento de los ODS.

1.5. Colaboraciones

Este TFG parte de un proyecto que originalmente se desarrolló para la asignatura de Desarrollo de Videojuegos 2D. Dicho proyecto se realizó junto a mis compañeros Francisco José Noguera Guijarro y Jorge Duart Marzo, centrándonos cada uno en una parte del videojuego, siendo yo el responsable de la implementación de los enemigos del juego. En cuanto a la parte artística, al no contar con artistas en nuestro equipo, todo el arte se obtuvo de terceros a través de plataformas especializadas.

En el proyecto original se consiguió implementar el nivel (incluyéndose su generación aleatoria), las mecánicas principales del jugador y tres enemigos. Dichos enemigos fueron el enemigo de ataque a distancia, el enemigo de ataque a melee y el jefe del primer nivel, la Abuela Boss. Dichos enemigos se implementaron siguiendo un patrón de comportamiento bastante básico, basado únicamente en perseguir y atacar al jugador.

Figura 1: Pantalla de créditos de Black Friday War

2. Estado del Arte

En este capítulo se analizarán títulos del sector que comparten características con el videojuego desarrollado y/o pertenecen al mismo género, realizándose una comparación y así encontrar lo que hace destacar a Black Friday War de la competencia. También se estudiarán las técnicas más utilizadas para la implementación de la IA en el desarrollo de videojuegos, para guiar la selección de la técnica utilizada para el desarrollo de este proyecto.

2.1. Análisis y comparativa de títulos similares

A continuación, se estudiarán los videojuegos del mercado que más se asemejan a Black Friday War, especialmente aquellos que se incluyen dentro de su mismo género: los Roguelike.

2.1.1. Definición del género Roguelike

Una definición rápida para el género, sería decir que es aquel cuyos videojuegos se asemejan al Rogue, título del que hablaremos más tarde. Una definición más concreta sobre el género sería la planteada durante la Interpretación de Berlín de 2008, en la que desarrolladores de este tipo de títulos establecieron las características que un videojuego debía cumplir para considerarse un Roguelike, distinguiendo entre factores de mayor valor y factores de menor valor [4][5].

Entre los factores de mayor valor encontramos:

- Generación aleatoria del escenario: El mundo del juego se genera de forma aleatoria para incrementar la rejugabilidad.
- Muerte permanente: Si el jugador muere durante la partida, debe volver a empezar desde el principio.
- Basado en turnos: Cada acción/movimiento del jugador se considera un turno.
- Basado en celdas: El mundo del juego se divide en celdas. El jugador y los enemigos ocupan siempre una celda.
- No modal: El movimiento, el combate y cualquier acción son accesibles en todo momento durante el videojuego.
- Complejidad: El título debe permitir que un mismo reto pueda tener un gran número de soluciones.
- Administración de recursos: El jugador debe administrar sus recursos limitados buscando usos para ellos.
- Hack ‘n’ slash: Una parte importante de los Roguelike es que el jugador debe luchar contra un gran número de enemigos hasta acabar con ellos.
- Exploración y descubrimiento: El jugador debe explorar cuidadosamente los niveles y descubrir cómo usar los objetos encontrados.

En cuanto a los factores de bajo valor encontramos:

- Un solo personaje: El jugador controla únicamente un personaje, que es el centro del juego. Se experimenta el mundo del juego a través del personaje y el juego termina cuando dicho personaje muere.
- Los monstruos son similares al jugador: Los enemigos del juego pueden realizar las mismas acciones que el jugador (usar ítems, habilidades, etc.).
- Desafío táctico: El jugador debe aprender las tácticas y estrategias para superar los retos del juego antes de conseguir grandes progresos.

- Gráficos ASCII: El mundo del juego se representa mediante caracteres ASCII.
- Mazmorras: Los Roguelike contienen mazmorras, es decir, niveles compuestos por habitaciones y pasillos.
- Números: Las estadísticas del jugador (vida, ataque, etc.) se muestran con números.

A pesar de ello, la mayoría de los juegos clasificados dentro del género Roguelike no llegan a cumplir todos los factores que acabamos de describir. Es por ello por lo que existe también otra definición para aquellos títulos que no satisfacen todos los factores, los llamados Roguelite. Aunque en la mayoría de las ocasiones, los términos Roguelike y Roguelite se suelen usar sin distinción alguna, sirva de ejemplo la plataforma Steam en la que podemos encontrar ejemplos de títulos que se clasifican en ambos géneros [4].

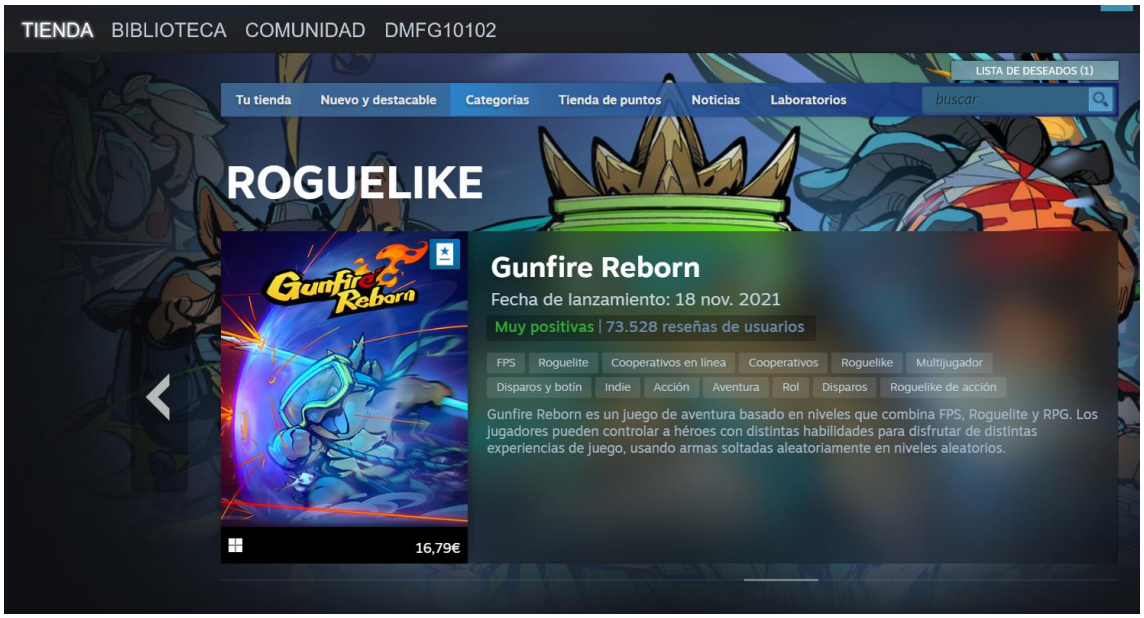


Figura 2: Vista de la categoría Roguelike en Steam, con un videojuego catalogado como Roguelike y Roguelite

2.1.2. Rogue

El videojuego Rogue es un juego de exploración de mazmorras basado en los juegos de rol del estilo Dungeons and Dragons. Fue publicado en 1980, convirtiéndose en el principal precursor del género Roguelike, llegando a dar nombre al mismo. Por las limitaciones técnicas de la época, se usaron caracteres ASCII para representar los gráficos del juego, aunque existen versiones posteriores que añaden plantillas gráficas y colores.

En Rogue, el jugador encarna a un aventurero que se interna en una peligrosa mazmorra en busca del Amuleto de Yendor. El objetivo del jugador es bajar a través de los distintos niveles de la mazmorra hasta llegar al Amuleto, tras lo que deberá ascender de nuevo al primer nivel para poder escapar al exterior, teniendo sumo cuidado en no morir, ya que la muerte del jugador es permanente. Los niveles de la mazmorra son generados de forma aleatoria en cada partida, ofreciendo un nuevo reto al jugador en cada ocasión.

A lo largo de la mazmorra, el jugador se irá enfrentando a enemigos que serán más fuertes conforme avance. Para vencerlos deberá hacer un buen uso de los distintos recursos que

encontrará a lo largo de la partida, como armas, pociones, trampas, comida, etc. Además, también deberá asegurarse de ganar experiencia para que su personaje se fortalezca. Rogue es un juego basado en turnos, con lo que el tiempo del juego no avanza hasta que el jugador no realice alguna acción como consumir un objeto, atacar, descansar, etc [6].

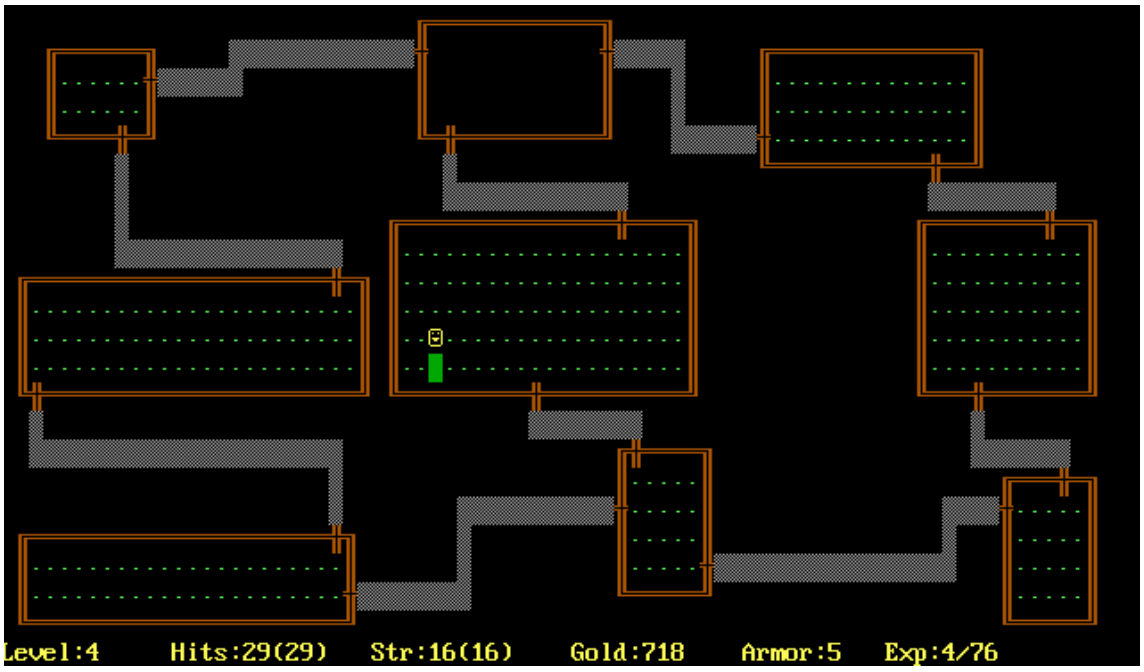


Figura 3: Rogue, Epyx Inc

2.1.3. Gauntlet

Gaunlet es un juego arcade y hack ‘n’ slash de exploración de mazmorras que, al igual que Rogue, se inspira en los juegos de rol del estilo Dungeons and Dragons. Fue publicado 1985 para las salas recreativas granjeándose un gran éxito, tras lo que se realizaron adaptaciones para otras plataformas e incluso secuelas, contando con un remake lanzado en 2014. Aunque Gauntlet tal vez no pueda considerarse un Roguelike de forma estricta según la Interpretación de Berlín, podemos encontrar en él muchas características en común con los títulos del género.

En Gauntlet, el jugador o jugadores (ya que la máquina original permitía jugar hasta a cuatro personas) encarnan a un aventurero que explora una mazmorra en busca de tesoros (una premisa bastante parecida a la de Rogue). Como en la mayoría de juegos de recreativas, el principal objetivo de los jugadores es llegar lo más lejos posible y obtener la mejor puntuación antes de morir. A diferencia de Rogue, los niveles de Gauntlet no son generados de forma aleatoria, contando la versión original con un total de cien niveles prediseñados, aunque el orden de aparición de los niveles se vuelve aleatorio tras el nivel ocho.

Los jugadores antes de iniciar el juego deben escoger un personaje entre los cuatro disponibles, cada uno con sus fortalezas y debilidades: el guerrero, el elfo, el mago y la valquiria. La clave para avanzar es utilizar bien las ventajas que ofrece el personaje escogido y, en el caso de jugar con más jugadores, coordinarse con ellos para suplir sus debilidades. A ser un juego en tiempo real (no se basa en turnos) los jugadores deben estar atentos para superar las hordas de monstruos que se encuentran en los niveles, bien evitándolos, matándolos a todos, destruyendo sus generadores para evitar que aparezcan más, etc. Los jugadores también deberán administrar de forma acertada los recursos que encontrarán a lo largo de la mazmorra, como son los

potenciadores, las pociones que permiten realizar encantamientos, las llaves que abren puertas o la comida, elemento indispensable para sobrevivir, ya que restaura los puntos de vida del jugador (los cuales no solo se reducen por el ataque de enemigos sino también con el paso del tiempo). A diferencia de Rogue, la muerte no es necesariamente "permanente", si aún quedan jugadores vivos o ha pasado poco tiempo desde la muerte del personaje, el jugador tenía la posibilidad de resucitar insertando una moneda en la máquina, elemento que convirtió a Gauntlet en uno de los juegos más rentables de las recreativas [7][8].

Figura 4: Gauntlet, Atari Games

2.1.4. The Binding of Isaac

No se puede hablar de los Roguelike actuales sin mencionar The Binding of Isaac, uno de los principales responsables de la popularidad que goza en la actualidad el género. The Binding Of Isaac es un título independiente de exploración de mazmorras con una fuerte inspiración en RPGs como The Legend Of Zelda. Fue lanzado originalmente en 2011 en Steam, obteniendo un gran éxito y popularidad, consiguiendo ediciones para otras plataformas y varias expansiones.

A diferencia de Rogue y Gauntlet, títulos en los que la historia apenas tenía importancia o era prácticamente inexistente, The Binding of Isaac cuenta con una historia compleja basada en el relato bíblico del Sacrificio de Isaac. En el juego, el jugador encarna a Isaac, un niño pequeño que, para evitar ser asesinado por su madre, quien cree haber recibido órdenes de Dios, decide esconderse en el sótano, dónde deberá enfrentarse a terribles monstruos para sobrevivir y escapar. El objetivo del jugador es simple: atravesar los niveles que componen el sótano sin morir.

Comparándolo con Rogue, podemos encontrar algunas similitudes, como la generación aleatoria de los niveles del sótano (o mazmorra) o la muerte permanente que te obliga a volver a empezar desde el principio, pero también encontramos grandes diferencias como es el tiempo real (no se basa en turnos) o la existencia de jefes, siendo necesario vencer al jefe de cada nivel para pasar al siguiente. Al igual que en Gauntlet, el jugador puede escoger entre varios personajes (representaciones alternativas de Isaac) que ofrecen ventajas y desventajas distintas, la gran mayoría de ellas bloqueadas al principio del juego, pudiendo desbloquearse completando logros durante las partidas, los cuales que no se pierden, aunque muera el jugador. Es decir, la muerte del jugador no llega a borrar todos sus progresos.

En cuanto al combate, este se basa en disparos, siendo el principal ataque de Isaac el lanzamiento de lágrimas. Para superar el juego, el jugador deberá luchar contra los monstruos que encuentre en las distintas salas que componen cada nivel, siendo imposible salir de ellas mientras queden enemigos vivos, lo que obliga al jugador a vencerlos, buscando las mejores estrategias para vencer a cada clase de enemigo. El jugador también se verá obligado a administrar sus recursos, como son los consumibles o las llaves, además de encontrar y equipar armas, accesorios y potenciadores para mejorar al personaje y tener mayores posibilidades de vencer a los enemigos más poderosos [9][10].

Figura 5: The Binding of Isaac, Nicalis Inc.

2.1.5. Enter The Gungeon

Enter The Gungeon es un Roguelike fuertemente influenciado por The Binding Of Isaac que lleva el combate con disparos a cotas mayores llegando a clasificarse también como un Bullet Hell. Fue lanzado en 2016 para ordenador y PS4 (posteriormente se lanzó también para Xbox One y Nintendo Switch) consiguiendo un gran éxito, sacándose expansiones y un spin-off.

En comparación con The Binding of Isaac, la historia de Enter The Gungeon es bastante simple y sirve más como justificación de la acción que como motivación para el jugador: en un planeta remoto existe la Gungeon o "Armamazmorra", una mazmorra que oculta un arma legendaria capaz de matar el pasado, lo que atrae a distintos aventureros que desean cambiar su pasado. El objetivo del jugador es atravesar los niveles de la mazmorra y conseguir el arma legendaria.

Al igual que en The Binding Of Isaac y Gauntlet el jugador debe escoger el personaje con el que jugará la partida, teniendo cada uno sus fortalezas, debilidades y su propio equipamiento inicial. En total, el jugador cuenta con cuatro personajes, conocidos como los Gungeonists o "Armamazmorristas", quiénes estarán disponibles desde el principio del juego (no hay ninguna necesidad de desbloquearlos) siendo estos el piloto, el marine, la convicta y la cazadora. También existe un quinto personaje, el cultista, que es utilizado en el modo cooperativo local por el segundo jugador.

Como en todos los Roguelike vistos, los niveles son generados de forma aleatoria en cada intento y, si el jugador muere, debe volver a empezar desde el principio, aunque al igual que en

The Binding Of Isaac existen logros que no llegan a perderse, principalmente a través de los encuentros con personajes en la mazmorra, que aparecerán posteriormente en la sala inicial del juego para hablar, iniciar retos, jugar minijuegos, etc. pudiendo incluso desbloquearse atajos que permiten al jugador saltarse niveles. Al igual que en The Binding Of Isaac, al entrar en una sala con enemigos, es imposible salir de ella sin terminar el combate, los cuales tienden a ser muy frenéticos y a llenarse de muchos proyectiles que habrá que evitar mediante coberturas o usando la "voltereta" en el momento apropiado. Cada nivel cuenta también con un jefe al que hay que derrotar antes de pasar al siguiente nivel.

Al avanzar por la Gungeon, los enemigos se van haciendo más y más fuertes, por lo que es indispensable que el jugador busque y administre bien los recursos del juego, escogiendo cómo usar los muchos consumibles, las armas disponibles (cada una con sus estadísticas y habilidades propias) con munición limitada, los accesorios (que añaden habilidades al jugador), las mejoras y el dinero, que permite comprar en las tiendas disponibles en la Gungeon, con el objetivo de mantener vivo al personaje y fortalecerlo [11][12].

Figura 6: Enter the Gungeon, Dodge Roll

2.1.6. Hades

Otro de los títulos que es necesario mencionar para entender los Roguelike actuales es Hades, un frenético RPG Roguelike centrado en la acción Hack 'n' slash que fue lanzado oficialmente en 2020, siguiendo a su lanzamiento de acceso anticipado de 2018. Hades cosechó un gran éxito entre crítica y público, llegando a ganar diversos premios.

A diferencia de Enter The Gungeon, la historia, basada en la mitología griega, es más compleja y además de justificar la acción trata de motivar al jugador para "engancharlo" al universo del juego. En Hades, el jugador controla a Zagreus, el hijo rebelde del dios Hades, señor del inframundo, que, cansado de los reinos de su padre, trata por todos los medios de escapar al mundo exterior. El objetivo del juego vuelve a ser el mismo de siempre en los videojuegos Roguelike: atravesar una mazmorra hasta alcanzar el final sin morir, encarnando el inframundo griego dicha mazmorra.

De nuevo, a cada intento los niveles del inframundo se generan de forma aleatoria y, al morir, el jugador debe volver a empezar desde el principio, aunque, al igual que en The Binding Of Isaac y Enter The Gungeon, no es una muerte permanente tan estricta, conservando progresos en forma de nuevas interacciones con personajes para conocer mejor la historia de juego o misiones. También conserva una serie de puntos que permiten fortalecer al personaje, ya sea mediante la mejora de sus habilidades, el desbloqueo de nuevas armas y la mejora de estas, lo que ensalza su componente rolero.

En cuanto al combate, si The Binding Of Isaac y Enter The Gungeon se centraban en los disparos, llegando el último a rozar el género Bullet Hell, Hades se centra en la acción cuerpo a cuerpo Hack 'n' slash del estilo de juegos como God Of War, en el que la clave no es solo encontrar la mejor forma de vencer a cada tipo de enemigo, sino también dominar las armas de Zagreus. De nuevo, cada vez que se entra en una sala de nivel, el jugador no puede salir de ella hasta vencer a todos los enemigos, y para pasar al siguiente nivel deberá vencer al jefe de nivel.

De nuevo, otra de las claves para enfrentarse a los cada vez más arduos retos es la de administrar bien los recursos que se te presentan, seleccionando bien el arma con el que jugarás la partida, escogiendo y dando buen uso a los potenciadores encontrados en los niveles y utilizando de forma correcta los consumibles [13].

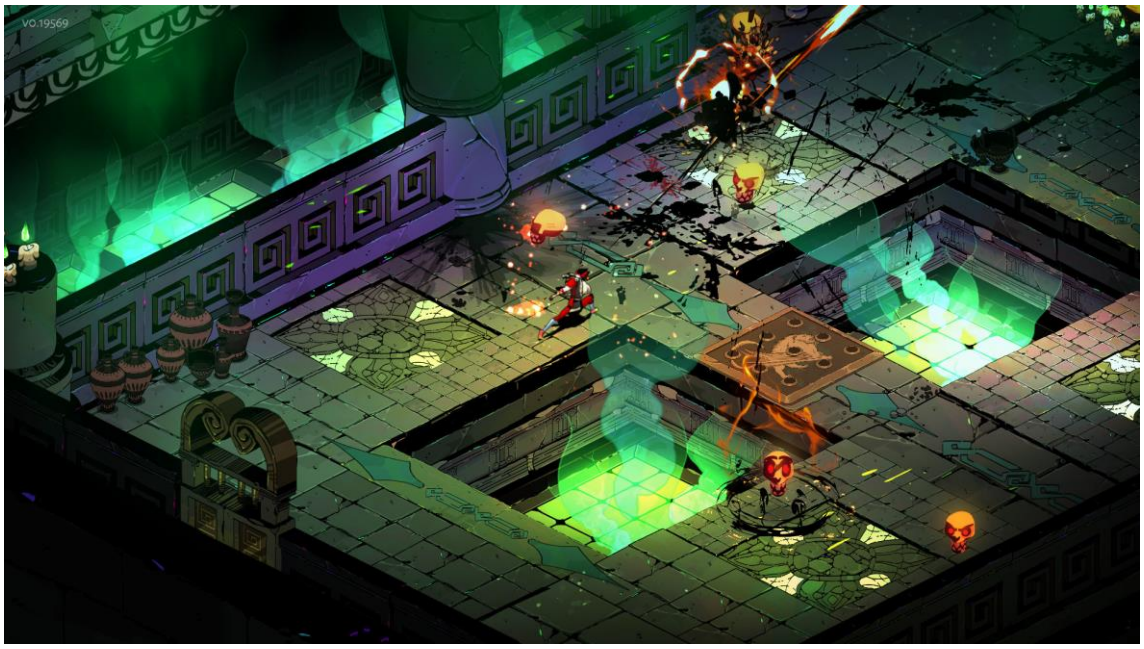


Figura 7: Hades, Supergiant Games

2.1.7. Black Friday War

Black Friday War es el videojuego que se ha desarrollado a lo largo de este proyecto. Este título es un Roguelike de acción con una fuerte inspiración en títulos como Enter the Gungeon o The Binding of Isaac.

El argumento del videojuego es bastante sencillo y funciona como una justificación de la acción. En Black Friday War se traslada al jugador a una sociedad futura sumida en la decadencia y el caos por culpa de las continuas crisis internacionales, lo que ha ocasionado un brutal encarecimiento del nivel de vida. En este contexto, el evento conocido como Black Friday se ha

convertido en la única esperanza de supervivencia de la gente por sus ofertas y descuentos, lo que ha provocado que los compradores sean capaces de cualquier cosa por adquirir la mejor oferta. En este escenario, el jugador encarna a la familia Remunson, que para sobrevivir un año más deciden internarse en el campo de batalla que se ha convertido el Black Friday.

El objetivo del juego es de nuevo atravesar una mazmora sin morir, siendo en esta ocasión un centro comercial la encarnación de la mazmorra. Cómo ya se ha mencionado, el jugador encarna a la familia Remunson, conformada por cuatro personajes: el padre, la madre, el hijo y la hija. A diferencia del resto de títulos analizados en los que el jugador debe seleccionar en cada partida el personaje con el que jugará, en Black Friday War el jugador controla en todo momento a la familia completa, que se mueven juntos en formación.

Cada personaje cuenta con sus ventajas y desventajas, además de una habilidad especial que el jugador puede activar en cualquier momento, cada una con su propio efecto (la habilidad del padre potencia el ataque, la habilidad de la madre ofrece invulnerabilidad temporal, la habilidad del hijo aumenta la velocidad y la habilidad de la hija es un ataque de área). El jugador se considera derrotado cuando todos sus personajes han sido derrotados, es decir, mientras aún quede un personaje con vida, el juego no termina.

En este caso, la muerte permanente es estricta: al morir el jugador se pierden todos los progresos conseguidos obligando al jugador volver a comenzar una nueva partida desde el principio. De nuevo, como en todos los Roguelike, en cada partida los niveles cambian, habiendo una nueva distribución de las salas y de los enemigos.

En cuanto al combate, este se basa en el uso de disparos y ataques cuerpo a cuerpo mientras se utilizan las habilidades de los personajes para obtener ventaja frente a los enemigos. Cómo siempre, es imposible salir de las salas hasta que no se haya derrotado a todos los enemigos. Para finalizar el nivel, el jugador debe vencer al jefe del nivel, pero antes debe buscar y reunir tres objetos que le darán el derecho a entrar en la sala del jefe. Dichos objetos representan las provisiones que la familia Remunson necesita para sobrevivir.

Por último, hay que mencionar que de nuevo una de las claves para enfrentarse a los retos del juego es administrar los recursos utilizados, seleccionando que accesorios potenciadores equipar a los personajes y cómo usar los ítems de recuperación soltados por los enemigos.

Figura 8: Black Friday War

2.1.8. Black Friday War en el género Roguelike

Una vez definido el concepto de Roguelike y visto algunos títulos, pasaremos a analizar cómo se enmarca nuestro proyecto, Black Friday War, dentro de los Roguelike. Para ello analizaremos qué factores establecidos por la Interpretación de Berlín cumple.

Estudiando las características de nuestro videojuego podemos ver que cumple un buen número de factores de la Interpretación de Berlín. Fijándonos en los factores de mayor valor, el primero que cumple sería la generación aleatoria de los niveles en cada partida. Aunque las salas que conforman los niveles son prediseñadas, el ensamblaje entre salas, los enemigos que aparecen en ellas y los ítems o accesorios que se pueden encontrar son completamente aleatorios. También habría que destacar la muerte permanente del jugador, que borra cualquier progreso al morir, obligándole a iniciar una nueva partida. Otro factor es el de que el juego sea no modal, es decir, que el jugador tenga acceso a todas las mecánicas del personaje desde el principio, factor que también se cumple, ya que el jugador tiene acceso desde el principio de la partida a los ataques y habilidades de los personajes. El factor de complejidad también podría considerarse cumplido, ya que el uso de los ataques y de las habilidades de los personajes permite vencer a los enemigos de distintas formas. También encontramos el factor hack 'n' slash cumplido, ya que el jugador se ve obligado continuamente a enfrentarse a grupos de enemigos, siendo imposible avanzar a nuevas salas si aún quedan enemigos vivos en la sala.

En cuanto al factor de administración de recursos, aunque no exista ningún inventario, podría decirse que se cumple en cuanto a la gestión del uso de los PH, o puntos habilidad que permiten utilizar las habilidades de los personajes, que, aunque puedan rellenarse recogiendo ítems, no siempre se asegura que se vayan a encontrar ítems de recuperación de puntos de habilidad. También podría considerarse cumplido si suponemos la selección y recolección de los ítems de recuperación de vida y de recuperación de PH como una especie de gestión del botín obtenido. Los enemigos al morir sueltan este tipo de ítems, que al ser recogidos afectan únicamente al personaje controlado por el jugador en ese momento y desaparecen al abandonar la sala, por lo que es necesario elegir con cuidado qué personajes del grupo deberían recuperar vida y/o recuperar PH, teniendo en cuenta que al salir de la sala los ítems sin usar desaparecerán. También,

si pensamos en la selección del accesorio que equipará cada personaje, teniendo en cuenta que cada personaje puede equiparse únicamente uno y que solo afectará al grupo cuando ese personaje sea controlado por el jugador, podríamos verlo como otra forma de administración de recursos.

Por último, se cumple el factor de exploración y descubrimiento, aunque de forma parcial. En Black Friday War el jugador se ve obligado a explorar el nivel para encontrar la sala del jefe, además de las llamadas salas de artículo, siendo necesario encontrarlas para obtener los tres artículos clave del nivel y así poder entrar en la sala del jefe. Por otra parte, la exploración también puede conllevar algunas recompensas, como el encuentro de salas del tesoro en las que se pueden encontrar accesorios potenciadores. En cuanto al "descubrimiento" de cómo usar los objetos del juego, es la parte del factor que no llega a cumplir Black Friday War, ya que el uso de los objetos es bastante intuitivo por su representación. Los factores de basar el juego en turnos o dividir el escenario en celdas no se cumplen.

En cuanto a los factores de prioridad baja, solo se cumple el de las mazmorras, ya que elBlack Friday War tiene lugar en un escenario de ese estilo, dividido en distintas salas, aunque argumentalmente el escenario del juego represente unos grandes almacenes. El resto de los factores no llegan a cumplirse, ya que no controlamos un único personaje, sino cuatro en una misma partida, no se muestran todas las estadísticas de los personajes (solo la vida y los usos de habilidad) ni se representan con números y tampoco se utilizan gráficos ASCII, usándose un estilo gráfico pixel art. En cuanto al factor de desafío táctico, aunque al principio sí que hay cierta dificultad para descubrir las mejores formas de vencer a los enemigos, al final pueden llegarse a estrategias que pueden ser útiles en todas las partidas, sobre todo a la hora de enfrentarse a los jefes finales.

Aunque apenas se cumplan los factores de valor bajo, al cumplir gran parte de los factores de alta prioridad, podemos incluir a Black Friday War dentro del género Roguelike.

2.1.9. Comparativa

Ahora que ya hemos analizado las características de los Roguelike, además de estudiar la relación de nuestro proyecto con dicho género, podemos pasar a hacer una comparación entre los títulos descritos y el desarrollado para este proyecto. La tabla siguiente describe las características principales de todos los títulos vistos, incluyendo el nuestro, en la que podemos observar las semejanzas y diferencias existentes entre ellos.

Tabla 1: Comparativa de videojuegos

	Rogue	**Gauntlet**	**The Binding Of Isaac**	**Enter The Gungeon**	**Hades**	**Black Friday War**
Niveles generados aleatoriamente	Sí	No	Sí	Sí	Sí	Sí
Varios personajes jugables	No	Sí	Sí	Sí	No	Sí
Muerte permanente	Sí	No	Sí	Sí	Sí	Sí
Basado en turnos	Sí	No	No	No	No	No
Exploración	Sí	Sí	Sí	Sí	Sí	Sí
Hack 'n' slash	Sí	Sí	Sí	Sí	Sí	Sí
Jefes de nivel	No	No	Sí	Sí	Sí	Sí

Administración de recursos	Sí	Sí	Sí	Sí	Sí	Sí
Multijugador	No	Sí	Sí	Sí	No	No

2.2. IA en los videojuegos

Uno de los aspectos más importantes de un videojuego es la aplicación de técnicas de Inteligencia Artificial, usualmente para la implementación de los NPC (Non Playing Characters o personajes no jugadores). El Parlamento Europeo define la Inteligencia Artificial como "la habilidad de una máquina de presentar las mismas capacidades que los seres humanos, como el razonamiento, el aprendizaje, la creatividad y la capacidad de planear" [15]. En el desarrollo de videojuegos, el uso de IA no tiene como objetivo conseguir replicar la inteligencia humana, sino "construir algoritmos que hagan que los personajes del juego parezcan humanos o animales", es decir, hacer que los personajes del juego parezcan inteligentes [16].

Además, al desarrollar la IA de un videojuego ha de tenerse en cuenta que el fin último del juego es que el usuario se divierta, siendo la clave de ello ofrecer al jugador un grado adecuado de desafío, es decir, hacer que el juego no sea ni muy fácil ni muy difícil, ya que un juego muy sencillo puede perder rápidamente el interés del jugador y un juego casi imposible cansará al jugador.

Para entenderlo mejor, pongamos como ejemplo el clásico videojuego Pac-Man y la IA de los fantasmas. Imaginemos que se implementa una primera versión de la IA que hace que los fantasmas sean fáciles de burlar y esquivar por Pac-Man, hasta tal punto que parece que lo evitan. Eso haría que el jugador fuese capaz de pasarse el juego sin apenas esfuerzo y en un juego que se gana siempre, ¿qué necesidad hay de jugar? Ahora imaginemos el caso contrario, que se implementa una segunda versión de la IA que cuenta con un algoritmo mejorado de Pathfinding que permite a los fantasmas alcanzar más rápidamente a Pac-Man y acorralarlo en apenas unos segundos, haciendo casi imposible que Pac-Man escape. Tras jugar un par de partidas, después de haber frustrado al jugador al ver que es imposible ganar, ¿para qué va a seguir jugando?

Por lo tanto, los desarrolladores no solo deben preocuparse por la corrección de la IA, sino que también deben asegurarse de ajustarla para que ofrezca un reto adecuado al jugador y así pueda disfrutar del videojuego.

2.2.1. Técnicas de IA en los videojuegos

Para el desarrollo de la IA en los videojuegos, se suelen usar diversas técnicas entre las que encontramos las siguientes [17]:

- Máquinas de estados finitos. Las máquinas de estados finitas o, en inglés, Finite State Machines (FSM) son, tal y como se menciona en Unity Artificial Intelligence Programming de Davide Aversa, "probablemente uno de los más simples, más usados y más debatidos modelos de IA y, para la mayoría de los juegos, representan la única técnica de IA". Las máquinas de estados consisten en un conjunto de estados conectados entre sí por transiciones, formando un grafo, como se puede ver en la Figura 9.

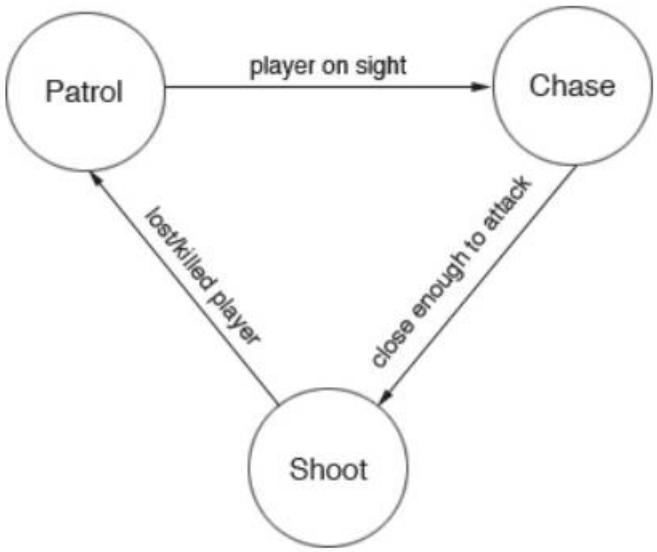


Figura 9. Ejemplo de máquina de estados sencilla para un enemigo, Unity Artificial Intelligence Programming

Entre títulos que hacen uso de las FSM encontramos Hollow Knight.

Figura 10: Hollow Knight, Team Cherry

- Machine Learning. El machine learning es un conjunto de técnicas basadas en el uso de algoritmos que aprenden de los datos, identifican patrones y toman decisiones sin programación o intervención humana. No es una técnica muy utilizada en el desarrollo de videojuegos, ya que el desarrollador es incapaz de controlar las salidas de la entidad, lo que se traduce en comportamientos impredecibles que pueden hacer que el juego deje de ser divertido, por lo que se suelen utilizar técnicas más predecibles como los Behavior Trees o las Máquinas finitas de estados. El machine learning es más utilizado para simulaciones o para el desarrollo de los denominados juegos serios.

  Un ejemplo de videojuegos que utilizan Machine Learning sería Middle-Earth: Shadow of Mordor y su secuela Middle-Earth: Shadow of War, observándose en el comportamiento del ejército orco, capaz de aprender y adaptarse ante los actos del

jugador. Esto se observa sobre todo en los capitanes orcos que, al combatir contra el jugador, pueden desarrollar fobias o fortalezas, e incluso provocar que el capitán sienta "odio" por el jugador y que en un futuro busque venganza.

Figura 11: Middle-Earth: Shadow of War, Monolith Productions

- Behaviour Trees. Otra de las técnicas utilizadas para encapsular el comportamiento de los personajes, popularizada por títulos como Halo 2. Un Behavior tree o BT se basa en una estructura en árbol, donde normalmente los nodos hoja representan acciones a ejecutar o condiciones a comprobar.

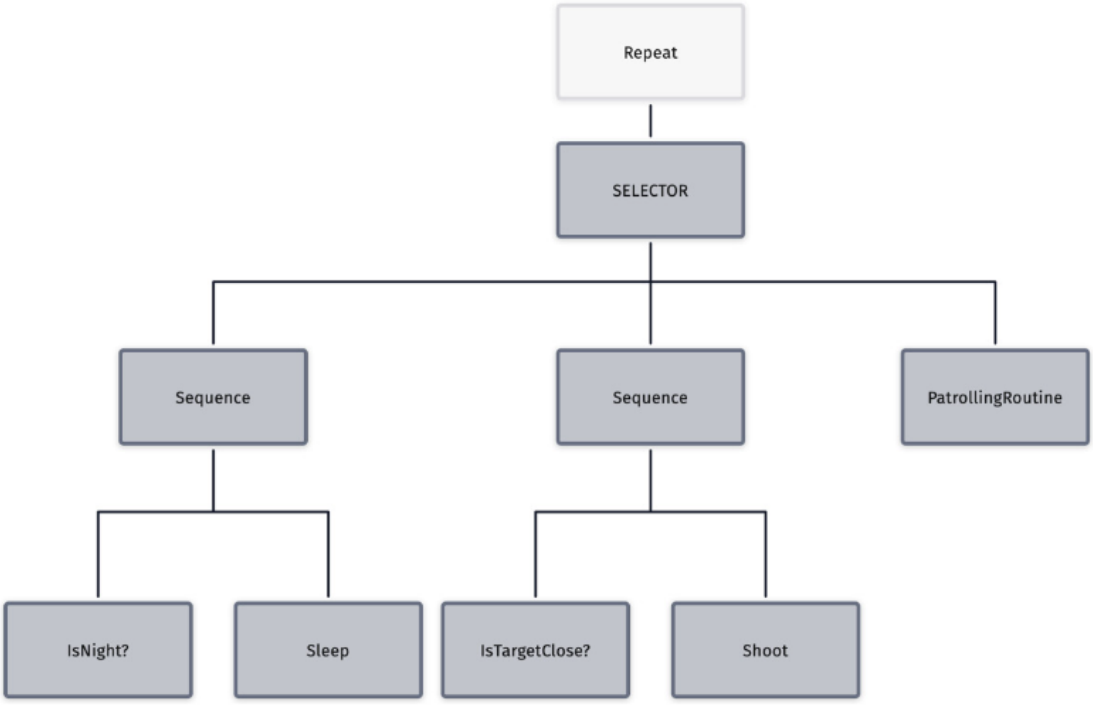


Figura 12: Ejemplo de Behavior Tree, Unity Artificial Intelligence Programming

Entre títulos que utilicen esta técnica encontramos Assassins Creed III, Tom Clancy's Splinter Cell: Blacklist o Project Zomboid, cuyo creador además dedicó un artículo a dicha técnica.

Figura 13: Project Zomboid, The Indie Stone

Tras estudiar detenidamente dichas técnicas, finalmente se decidió usar Behavior Trees para la implementación de la IA de los enemigos de Black Friday War. Esta decisión se debe a que los Behavior Trees ofrecen ventajas referentes a su mantenimiento y reusabilidad, además de ser una técnica sencilla e intuitiva.

2.2.2. Behavior Trees

Como ya hemos mencionado anteriormente, los árboles de comportamiento o, en inglés, Behavior Trees (BT para abreviar) es una técnica de IA en la que el comportamiento de la entidad se modela mediante una estructura con forma de árbol.

Los BT surgen como una alternativa a las FSM (máquinas de estado) que, a pesar de ser una técnica simple e intuitiva, cuentan con el inconveniente de que, al crecer en número de estados y transiciones, se vuelve más difícil mantenerlas. Algunos autores ven las BT como una forma alternativa de representar las FSM, siendo las BT más rápidas, ofreciendo reusabilidad y con un mantenimiento más sencillo.

Un BT es un árbol jerárquico de nodos que controla el flujo de comportamiento de la entidad. Cuando los nodos de un BT son ejecutados, devuelven como salida los estados success, failure o running. Se pueden distinguir tres tipos de nodos:

- Tareas. Nodos hoja del árbol, normalmente representan condiciones a comprobar o acciones del personaje.
- Decoradores. Nodos del árbol con un solo hijo, se utilizan para modificar la salida de su hijo. Sirva de ejemplo el decorador inversor, el nodo que invierte el valor devuelto por su hijo, es decir, si el hijo devuelve success, este devuelve failure y viceversa.
- Compuestos. Nodos con varios hijos que sirven para dirigir el flujo de ejecución del nodo. Los nodos compuestos más conocidos y utilizados son el nodo selector y el nodo sequence o secuencia.
  El nodo selector ejecuta a sus hijos hasta que uno devuelva success, ejecutándolos en orden comenzando desde el hijo que se encuentra más a la izquierda hasta el que se encuentra más a la derecha. Cuando uno de los hijos devuelve success, termina de ejecutar

a sus hijos y devuelve success. En el caso de que ninguno devuelva success y todos devuelvan failure, al terminar de ejecutar todos los hijos devolvería failure.

En cuanto al nodo sequence, su comportamiento es el contrario al del selector. Al igual que el selector, ejecuta sus hijos desde el que se encuentra más a la izquierda hasta el último a la derecha, finalizando la ejecución cuando un hijo devuelve failure, tras lo que devuelve failure. En el caso de que ningún hijo devuelva failure, y todos devuelvan success, tras ejecutar todos los hijos, el nodo sequence devolverá success.

Para comprender mejor el funcionamiento de los BT y sus nodos analizaremos el ejemplo visto en el punto anterior. Dicho BT representa el comportamiento de un robot patrulla enemigo en un videojuego. El comportamiento de dicho robot es el siguiente: si es de noche, el robot duerme, si es de día el robot realiza patrulla hasta que su objetivo (el jugador) se encuentra cerca, en cuyo caso le dispara.

Fijémonos en que, para representar dicho comportamiento, se ha usado un selector con tres hijos: dos nodos sequence (uno para hacer dormir al robot y otro para atacar al objetivo) y el nodo hoja PatrollingRoutine (que representa la rutina de comportamiento). Fijémonos en el orden en el que aparecen dichos nodos. Al diseñar un BT es indispensable escoger con cuidado el orden de los nodos, ya que por la forma en que funcionan los nodos compuestos, puede haber nodos que no se ejecuten. En este caso, en el caso de que fuera de noche, solo se ejecutaría el nodo sequence que hace dormir al robot, y el resto de los nodos no se ejecutarían, que es justo el comportamiento deseado: el robot cuando es de noche solo duerme, no hace nada más. En cambio, si hubiéramos puesto, por ejemplo, el sequence de atacar a la izquierda del de dormir, aunque fuera de noche (que es cuando toca que el robot duerma) atacaría al objetivo cuando se encontrara en rango y no se ejecutarían los nodos de dormir, es decir, atacaría de noche, cuando en realidad el comportamiento deseado es que ataque únicamente de día. De ahí la importancia del orden de los nodos hijos de los compuestos, ya que usar un determinado orden u otro puede afectar en gran medida al comportamiento final que mostrará la entidad.

Si observamos el primer nodo sequence (el de dormir), vemos que cuenta con dos nodos hoja: el nodo IsNight? y el nodo Sleep. El nodo IsNight? es un ejemplo de nodo condición, que en este caso comprueba si es de noche, mientras el nodo Sleep representa la acción de dormir. Fijémonos cómo se ha utilizado el orden de los hijos del sequence para conseguir el comportamiento deseado. En primer lugar, se ejecuta el nodo a la izquierda, IsNight?, comprobándose si es de noche, devolviendo success en caso afirmativo, con lo que el sequence seguiría ejecutándose y ejecutaría Sleep, haciendo dormir al robot, con el sequence devolviendo success al ejecutar todos sus hijos. Si no fuera de noche, IsNight? devolvería failure con lo que el sequence no llegaría a ejecutar sleep y devolvería failure. Lo mismo sucede con el segundo sequence, que cuenta con el nodo condición IsTargetClose? para comprobar si el objetivo se encuentra cerca, en cuyo caso se ejecutará el nodo Shoot, que representa la acción de disparar.

Para entenderlo mejor, revisemos una posible ejecución del BT, aquella en la que el robot se pone a patrullar porque no es de noche ni el objetivo se encuentra cerca. Se inicia la ejecución y el Selector empieza a ejecutar a sus hijos, comenzando por el primer nodo sequence. Dicho nodo ejecutará en primer lugar IsNight?, que devolverá failure al ver que es de día, con lo que el sequence finalizará su ejecución sin llegar al nodo Sleep y devolverá failure. Como el primer nodo hijo del selector devuelve failure, el selector ejecutará su siguiente hijo, el segundo nodo sequence. Este nodo ejecutará IsTargetClose? que devolverá failure al no estar el objetivo cerca, con lo que, al igual que el primer sequence, finalizará su ejecución sin llegar al nodo Shoot. Al

devolver el segundo sequence también failure, finalmente el selector ejecutará el último nodo, PatrollingRoutine, que ejecutará la rutina de patrulla del robot, devolviendo success, terminando la ejecución del selector.

Ahora bien, como hemos dicho anteriormente, una de las ventajas que ofrecen los BT es la reusabilidad. Esto se refiere a que los nodos creados para una BT pueden volver a utilizarse en la construcción de nuevos BT. Imaginemos que queremos crear una nueva versión del robot que realice nuevas acciones, pero queremos que el robot siga durmiéndose cuando sea de noche. Para la creación del nuevo BT no es necesario volver a crear los nodos IsNight? y Sleep, ya fueron creados para la primera versión del BT por lo que pueden volver a usarse en el BT del nuevo robot.

Además, los BT permiten hacer uso de BTs anidados, es decir, hacer uso de BTs dentro de otros BTs. Imaginemos que para el caso que acabamos de ver del robot, el comportamiento de patrulla fuera demasiado complejo para representarlo únicamente en un nodo. Dicho comportamiento podría modelarse mediante un BT que podría anidarse al BT del robot e incluso también en otros BT de entidades en las que se desee dicho comportamiento.

En conclusión, la técnica de los árboles de comportamiento o Behavior Trees es una solución interesante frente a otras técnicas populares en el desarrollo de videojuegos como son las máquinas de estado. Permiten modelar los comportamientos de una forma sencilla e intuitiva, ofreciendo además una gran reusabilidad que facilita la construcción de nuevos árboles de comportamiento [17][18].

3. Análisis del problema

Para este proyecto, el problema a resolver es el desarrollo de un videojuego Roguelike, concretamente la implementación de los distintos tipos de enemigos.

3.1. Especificación de requisitos

Como en todo proyecto software, lo primero es especificar los requisitos del proyecto. Para este proyecto se ha decidido ordenar los requisitos en Unidades de Trabajo (UT). En las metodologías ágiles, las UTs representan tareas del contexto del proyecto o elementos específicos que afectan al producto desarrollado (requisitos, mejoras, etc.) [19].

A su vez, para cada UT se han ido definiendo Pruebas de Aceptación (PA) con las que comprobar y validar cada requisito. Las PA describen escenarios específicos y una serie de pasos a seguir para obtener un determinado resultado, sirviendo para demostrar el cumplimiento de un requisito [20]. Utilizaremos las PA para validar cada UT con efecto directo sobre el producto.

3.1.1. Unidades de trabajo

Las Unidades de Trabajo del proyecto pueden visualizarse en las siguientes tablas, distinguiéndose entre las UT con un efecto directo sobre el videojuego, que tendrán asignadas a su vez un conjunto de PA, y las UT que representan tareas dentro del contexto de trabajo.

Tabla 2: Unidades de Trabajo con efecto directo sobre el producto

Efecto directo sobre el producto		
Código	**Nombre**	**Descripción**
MeleeSandbox	Enemigo Melee Sandbox	Versión incompleta, sin integrar en el juego final, del Enemigo Melee
DistanciaSandbox	Enemigo Distancia Sandbox	Versión incompleta, sin integrar en el juego final, del Enemigo Distancia
SanadorSandbox	Enemigo Sanador Sandbox	Versión incompleta, sin integrar en el juego final, del Enemigo Sanador
AnimadorSandbox	Enemigo Animador Sandbox	Versión incompleta, sin integrar en el juego final, del Enemigo Sanador
LímiteAnimado	Limitar del tiempo para el estado Animado	Poner un límite de tiempo al estado Animado de los enemigos para evitar una dificultad demasiado alta
EmbestidaSandbox	Enemigo Embestida Sandbox	Versión incompleta, sin integrar en el juego final, del Enemigo Embestida
AbuelaSandbox	Abuela Boss Sandbox	Versión incompleta, sin integrar en el juego final, de la Abuela, jefe del primer nivel

MadreHijoSandbox	Madre e Hijo Boss Sandbox	Versión incompleta, sin integrar en el juego final, de la Madre e Hijo, jefe del tercer nivel
DistanciaFinal	Enemigo a Distancia Final	Versión del Enemigo a Distancia integrada en el juego
MeleeFinal	Enemigo a Melee Final	Versión del Enemigo a Melee integrada en el juego
SanadorFinal	Enemigo Sanador Final	Versión del Enemigo Sanador integrada en el juego
AnimadorFinal	Enemigo Animador Final	Versión del Enemigo Animador integrada en el juego
EmbestidaFinal	Enemigo Embestida Final	Versión del Enemigo Embestida integrada en el juego
AbuelaFinal	Abuela Boss Final	Versión de la Abuela Boss integrada en el juego
NerdFinal	Nerd Boss Final	Versión del Nerd Boss integrada en el juego
MadreHijoFinal	Madre e Hijo Boss Final	Versión de la Madre e Hijo Boss integrada en el juego
CambioNivel	Sistema de cambio de nivel	Implementación del cambio de nivel al morir el jefe de nivel
AnimEfect	Animaciones y efectos de los enemigos	Finalizar las animaciones y efectos pendientes que no se hayan realizado en las otras UT
Retaguardia	Agrupar los apoyos en la retaguardia de los atacantes	Los Enemigos de Apoyo (Sanadores y Animadores) deben mantenerse detrás de los Enemigos Atacantes respecto del jugador

Tabla 3: Unidades de Trabajo relacionadas con el contexto de trabajo

Tareas en el contexto de trabajo
Seleccionar técnica de IA para modelar el comportamiento de los NPC
Búsqueda de librerías de BTs
Preparación del proyecto de prueba
Seleccionar librería para el uso de BTs
Seleccionar técnica de IA para el movimiento de los NPCs y Pathfinding
Modelar comportamiento del Enemigo a Melee
Modelar comportamiento del Enemigo a Distancia
Modelar comportamiento del Enemigo Sanador
Modelar comportamiento de Abuela Boss
Modelar comportamiento del Enemigo Animador
Modelar comportamiento del Enemigo Embestida
Modelar comportamiento del Nerd Boss
Modelar comportamiento de la Madre e Hijo Boss
Investigar aceleración e inercia en el movimiento de los personajes

Revisar configuración de los enemigos
Encuesta para Testers
Subir el videojuego a Itch.io

3.2. Modelado conceptual

Ahora que ya están establecidas las distintas Unidades de Trabajo y sus Pruebas de Aceptación, ha de buscarse una forma de representar de forma más visual como será la solución. Por una parte, para modelar el comportamiento de los distintos enemigos lo más sencillo es usar grafos para representar sus Behavior Trees.

Por otra parte, para la representación del sistema y sus componentes contamos con el estándar UML, el lenguaje de modelado más popular y conocido, que permite modelar desde la relación entre clases y módulos del sistema hasta sus casos de uso.

3.3. Análisis del marco legal

Uno de los puntos importantes a analizar es el marco legal del videojuego desarrollado para este proyecto, sobre todo en todo lo referente a la propiedad intelectual.

En cuanto a la autoría de Black Friday War, esta se comparte entre tres personas: Diego Manuel Fuentes García (responsable y autor del proyecto que se recoge en esta memoria), Francisco José Noguera Guijarro y Jorge Duart Marzo, con los que se desarrolló el proyecto original del que parte el actual. Los nombres de los tres autores del título aparecen en el GDD, que se encuentra disponible en los anexos, y en los créditos del juego.

En cuanto a las licencias de los recursos utilizados, la principal herramienta utilizada ha sido Unity. Dicho motor, tal y como se puede ver en su página oficial, cuenta con distintos planes o licencias, entre dichos planes encontramos el plan personal, que ha sido el plan seleccionado para el desarrollo del proyecto. Dicho plan es gratuito, dirigido a individuos o pequeñas empresas, otorgando el acceso a la versión más reciente de Unity, pero ha de cumplirse el requisito de que los ingresos del usuario o entidad sean inferiores a cien mil USD.

A parte de las herramientas, al no contar con artistas propios, tanto los sonidos como el arte se han obtenido de terceros. En lo referente al apartado visual, el arte tanto de los personajes como de los objetos, símbolos o escenarios se obtuvo de páginas con recursos gratuitos dirigidos al desarrollo de videojuegos: OpenGameArt e Itch.io. Dichos recursos se enmarcan en licencias Creative Commons, habiendo recursos con licencia CC0 (dominio público) y CC-BY. En cuanto a la música y los sonidos, todas las partituras y efectos de sonido se han obtenido de packs gratuitos disponibles en la Unity Asset Store, en la que se pueden encontrar muchos tipos de elementos para el desarrollo de videojuegos (arte, sonidos, plugins, scripts…) tanto gratuitos como de pago, permitiendo su fácil y rápida integración en los proyectos de Unity. Los recursos utilizados se han listado en la página de Itch.io de Black Friday War, acreditando a los autores que fuera necesario.

En cuanto a la pantalla del menú principal, el dibujo de fondo se realizó con una conocida herramienta de generación de imágenes mediante IA: Midjourney.

3.4. Solución original

Cómo ya se mencionó anteriormente, en el proyecto original se llegaron tres enemigos: dos normales (Enemigo Distancia y Enemigo Melee) y un jefe (la Abuela). El Enemigo Distancia y

el Enemigo Melee se caracterizan por sus ataques. El Enemigo Distancia lanza proyectiles al jugador mientras que el Enemigo Melee ataca cuerpo a cuerpo al jugador. La Abuela cuenta con ataques a distancia, basados en lanzar proyectiles, y un ataque a cuerpo a cuerpo en el que empuja al jugador.

La implementación de estos enemigos se basó en la aplicación de la técnica de las Máquinas de Estado. Las Máquinas de Estado utilizadas fueron muy sencillas y se basaron en el uso de una directiva switch. El comportamiento de estos enemigos es muy simple y básico, basándose únicamente en perseguir al jugador para atacarle hasta ser vencidos, sin llegar a haber interacciones entre los enemigos. En esencia, las Máquinas de Estados de los enemigos siguieron este modelo.

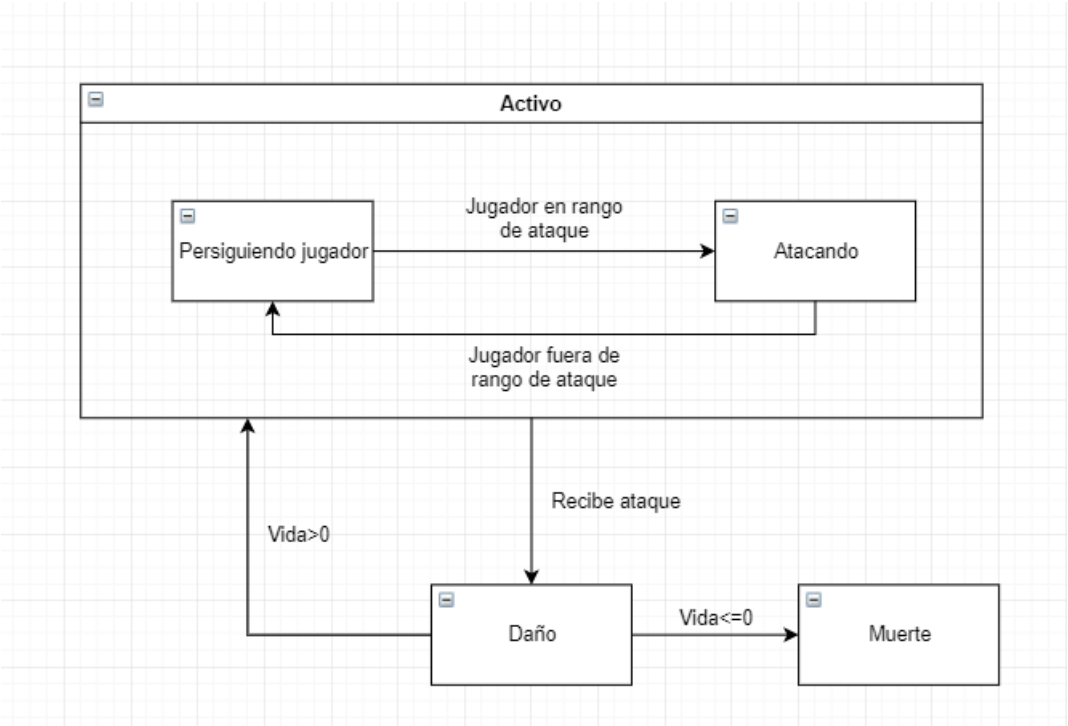


Figura 14: Máquina de estados básica de los enemigos

La principal desventaja de la implementación basada en Máquinas de Estados está ligada al mantenimiento de estas. Al crecer en estados cada vez se hace más difícil administrarlas, dificultando la modificación de estas. En el caso de que se quiera añadir un nuevo comportamiento a un enemigo ya definido, lo más probable es que sea necesario agregar nuevos estados que tendrán que conectarse a los nuevos, lo que se traducirá en tener que hacer modificaciones en las conexiones entre los estados, cuya dificultad dependerá de la complejidad de la Máquina de Estados.

En lo que respecta al movimiento de los enemigos, este se consiguió mediante la manipulación directa del componente Transform del enemigo, dirigiéndolo en línea recta hacia el jugador. Aunque es una forma simple y directa de abordar el movimiento de los enemigos, no es óptima, ya que el enemigo podría quedarse atascado intentando atravesar una pared para alcanzar al jugador.

En cuanto al control de los enemigos, las salas que componen el nivel cuentan con un componente controlador, el SalaControlador o HallController, que es el responsable de la

generación de los enemigos, usando para ello unos componentes Spawners para generar a cada enemigo. El HallController lleva un registro de los Spawners de la sala y, cuándo el jugador entra en dicha sala, los activa, generando una oleada de enemigos. El control de la generación del jefe de nivel sigue esta misma pauta, solamente que en este caso la sala solo cuenta con un Spawner, el del jefe, y la oleada solo consta de un enemigo.

En lo que respecta al cambio de nivel, en el proyecto original esto se tradujo únicamente en reiniciar por completo todo el nivel al vencer al jefe o al morir el jugador, sin llegar a guardar ningún registro del progreso del jugador en el caso de que este completara el nivel, volviendo a generarse una nueva distribución de las salas y de los enemigos, pero con el mismo jefe al final.

3.5. Solución propuesta

La solución propuesta para este proyecto es una actualización del sistema de enemigos que se desarrolló en el proyecto original. En dicha actualización, además de adaptar los tres Enemigos que se implementaron para el proyecto original (Enemigo Melee, Enemigo Distancia y Abuela) se añaden nuevos tipos, de tal forma que se pueden distinguir tres categorías: los enemigos atacantes, los enemigos de apoyo y los jefes de nivel.

Los enemigos atacantes, como dice el nombre, son los Enemigos que atacan al jugador. Existen tres tipos dependiendo del tipo de ataque: el Enemigo Distancia ataca lanzando proyectiles al jugador, el Enemigo Melee golpea cuerpo a cuerpo al jugador y el Enemigo Embestida carga contra el jugador embistiéndolo.

Los enemigos de apoyo no atacan al jugador, sino que intentan ayudar a los enemigos atacantes en su lucha. Se distinguen dos tipos: el Sanador, que rellena la vida del enemigo atacante, y el Animador, que otorga una mejora temporal a las estadísticas del enemigo atacante.

Los jefes de nivel son los Enemigos finales a los que el jugador se enfrenta al final del nivel para pasar al siguiente. En total hay tres (uno por nivel): la Abuela (jefe del primer nivel), el Nerd (jefe del segundo nivel) y la Madre e Hijo (jefes del tercer nivel).

Cada vez que el jugador inicie una partida, el jugador explorará las salas que componen el nivel, dónde se generarán de forma aleatoria enemigos atacantes y enemigos de apoyo. El jugador no podrá salir de la sala hasta acabar con todos los enemigos.

Tras conseguir los tres artículos clave del nivel, la puerta de la sala del jefe de nivel se abrirá, y podrá entrar en ella para dar inicio al combate contra el jefe. Al igual que con los combates con enemigos normales (atacantes y apoyos), el jugador no podrá salir de la sala mientras el jefe esté vivo. Al matar al jefe, el jugador pasará al siguiente nivel, dónde deberá vencer a un nuevo jefe.

Para ello se tendrá que realizar modificaciones sobre el sistema de control de los enemigos y de las salas, además de hacer posible que se pueda cambiar de nivel.

3.5.1. Control de niveles

En Black Friday War el nivel se considera finalizado cuando el jefe de nivel es derrotado. En total habrá tres niveles, contando cada uno con un jefe. Cuando el jefe sea derrotado, aparecerá un mensaje de victoria y, tras unos segundos, se iniciará el siguiente nivel. En el caso de que ese fuera el último nivel, se regresará a la pantalla de inicio del juego.

Para ello es necesario llevar un control sobre la cantidad de niveles que componen el juego y conocer el nivel en el que se encuentra el jugador en todo momento. También será necesario que

al cambiar de nivel no se reinicien todas las estadísticas del jugador y, que al menos, al iniciar un nuevo nivel se mantengan los personajes caídos, es decir, si en el primer nivel pierdes al padre, al iniciar el siguiente nivel el jugador solo contará con los personajes de la madre, el hijo y la hija.

Para conseguir esto, habrá que agregar un componente, que podríamos llamar SceneController, que lleve el control de los niveles. Dicho componente sabrá cuántos niveles componen el juego y el nivel en el que está el jugador. También será este componente el responsable de finalizar el nivel cuándo el jugador derrote al jefe.

Cuando el jugador venza al jefe, el SceneController comprobará si era el último nivel y, si es así, tras el mensaje de victoria el juego finalizará y se regresará a la pantalla de inicio. En el caso de que aún queden niveles, el SceneController actualizará la información referente a los niveles e iniciará el siguiente nivel. Para ello simplemente volverá a reiniciar el nivel, generándose una nueva distribución de las salas y de los enemigos, cambiando el jefe de nivel. El SceneController mantendrá algunas de las estadísticas del jugador y moverá al jugador a la sala inicial del nivel.

Para que esto funcione, el SceneController debe mantenerse en todos los niveles.

3.5.2. Enemigos

3.5.2.1. Enemigos normales

Cómo ya se ha dicho, al plantel de enemigos normales (que no son jefes) del proyecto original se añadirán nuevos tipos: un nuevo enemigo de ataque (el Enemigo Embestida) y enemigos de apoyo (el Enemigo Sanador y el Enemigo Animador). La principal novedad en lo que respecta a los enemigos es que habrá interacciones entre ellos y cierta aleatoriedad en su comportamiento.

Los enemigos atacantes (Enemigo Distancia, Enemigo Melee y Enemigo Embestida) seguirán persiguiendo y atacando al jugador, pero cuándo reciban daño al tener su vida reducida a menos de la mitad de la vida máxima, habrá un cincuenta por ciento de probabilidades de que entren dentro del llamado estado asustado. Dicho estado simula que el enemigo se ha acobardado al ver que está a punto de morir, por lo que dejará de atacar y comenzará a alejarse del jugador, es decir, huye. Dicho estado tendrá un límite de tiempo y, cuándo este sea alcanzado, el enemigo dejará el estado asustado y volverá a atacar al jugador, como si hubiera recuperado su valor.

Por otra parte, encontramos los enemigos de apoyo, enemigos que no tienen la capacidad para atacar al jugador y cuya principal función es ayudar a los enemigos atacantes. Estos enemigos se mantienen en todo momento tras los enemigos atacantes para evitar los ataques del jugador. Cuando un enemigo de apoyo detecta a un enemigo asustado, el apoyo y el enemigo asustado comienzan a acercarse y, al estar lo suficiente cerca, el apoyo comienza a "ayudarle". La “ayuda” que brinda el apoyo dependerá de que tipo de apoyo es.

En el caso del Enemigo Sanador, este sanará al enemigo asustado haciendo que recupere paulatinamente la vida, finalizando la sanación cuando el enemigo asustado tenga su vida al máximo o cuando uno de los dos reciba un ataque. Cuando se finaliza la sanación, el Enemigo sanador regresa a la retaguardia y el enemigo asustado abandona el estado asustado.

En cuanto al Enemigo Animador, este aplica una potenciación temporalmente al enemigo asustado, agregándole una bonificación de velocidad y ataque. Para que el Enemigo Animador aplique la potenciación sobre el enemigo asustado, debe pasar un tiempo determinado sin que ninguno de ellos reciba daño. Cuando haya pasado el lapso necesario, el Enemigo Animador volverá a la retaguardia y el enemigo asustado abandonará el estado asustado para regresar al

ataque estando potenciado. En el caso de que reciban daño antes de que pase el tiempo necesario, el Enemigo Animador regresará a la retaguardia y el enemigo asustado dejará el estado asustado para volver a atacar al jugador, pero sin potenciación alguna.

Cuando un enemigo ha sido potenciado por el Enemigo Animador, se considera dentro del estado animado. Al ser una potenciación temporal, el estado animado finalizará pasado un tiempo, recuperando su velocidad y ataque normal. Mientras un enemigo se encuentra dentro del estado animado, será imposible que entren en el estado animado. De esta forma, cuando un enemigo es encuentra en el estado animado, se simula que se encuentra motivado y capaz de cualquier cosa.

Cuando todos los enemigos atacantes son derrotados, los enemigos de apoyo huyen del jugador.

Respecto al Enemigo Distancia, aparte de los cambios referentes a las interacciones con los enemigos de apoyo y los nuevos estados, también contará con una pequeña modificación en su ataque. La condición de que el jugador se encuentre en su rango de ataque ya no es la única para que el Enemigo pueda realizar su ataque, ahora también debe comprobarse que exista una trayectoria limpia, es decir, que no haya ningún obstáculo (objetos del entorno) entre el jugador y el Enemigo. De esta forma evitamos que el Enemigo Distancia lance proyectiles cuando es imposible que dichos proyectiles alcancen al jugador.

En cuanto al Enemigo Embestida, como dice el nombre, se caracteriza por lanzarse contra el jugador para embestirlo. Cuando detecta al jugador, si el Enemigo no tiene obstáculos cerca ni tampoco otros enemigos, si existe una trayectoria limpia entre el Enemigo y el jugador en la que no haya objetos del entorno ni otros enemigos, el Enemigo inicia su embestida. La embestida es en línea recta hacia la posición en la que se detectó el jugador y, al alcanzar al jugador, además de dañarle, lo empuja. La embestida también puede fallar y golpear a otro enemigo, dañándolo.

Cuando el Enemigo Embestida alcanza con su ataque al jugador o a un enemigo, inicia una retirada. La retirada es bastante similar al estado asustado, ya que el Enemigo se aleja del jugador, pero no puede interactuar con los enemigos de apoyo y la retirada puede terminar de dos formas: cuando se alcanza el límite de tiempo de la retirada o el Enemigo es atacado, regresando otra vez al ataque. En el caso de que la embestida alcance un obstáculo, cómo una pared, el Enemigo se quedará aturdido, siendo incapaz de moverse hasta que se alcance el límite de tiempo para el aturdimiento o reciba un ataque, regresando otra vez al ataque o entrando en el estado asustado.

3.5.2.2. Jefes

Respecto a los jefes de nivel, se añaden dos nuevos: el jefe del segundo nivel (el Nerd) y el jefe del tercer nivel (Madre e Hijo).

3.5.2.2.1. Jefe del primer nivel

El jefe del primer nivel apenas cambia respecto a la primera versión del proyecto, manteniéndose la persecución constante del jugador, sus ataques a distancia y su ataque a cuerpo a cuerpo con empujón. También se mantiene el estado furia, que se activa cuando la vida del jefe es reducida a la mitad. Al encontrarse en ese estado, sufre un aumento de velocidad al moverse, sus proyectiles son más rápidos y hacen más daño, además de verse modificado su patrón de ataque.

3.5.2.2.2. Jefe del segundo nivel

En lo que respecta al Nerd, debemos decir que su movimiento se basa en desplazarse entre una serie de puntos clave colocados en los extremos de la estancia. Entre dichos puntos existen un conjunto de puntos intermedios entre los que se mueve de forma semi-aleatoria para alcanzar uno de los puntos clave o puntos destino. Cada vez que alcanza un punto destino, se espera un tiempo antes de moverse al siguiente.

Los ataques del Nerd se basan en el lanzamiento de drones. Dichos drones tendrán dos variantes: una normal y otra especial.

Los drones normales serán lanzados por el Nerd mientras se mueve o se prepara para moverse. Dichos drones son lanzados contra el jugador cuando el Nerd se mueve o mientras espera para moverse hacia un nuevo punto clave. También son lanzados cuando el Nerd llega a un punto clave, en cuyo caso se genera una hilera de drones en el extremo de la estancia en el que se encuentra el Nerd y son lanzados contra el otro extremo. Dichos drones irán en línea recta hasta su objetivo, desapareciendo al alcanzar un objeto del entorno, al jugador (en cuyo caso dañan al jugador) o son alcanzados por un ataque.

En cuanto a los drones especiales, estos son generados cuándo la vida del Nerd es menor o igual a la mitad de la vida máxima, es decir, cuándo se encuentra en estado furia. En este caso el estado furia solo agrega este ataque, en el que el Nerd va generando de forma periódica drones especiales. Estos drones no desaparecen al golpear al jugador o un obstáculo, sino que van cambiando de dirección de movimiento como si rebotaran. Además de dañar al jugador por contacto, estos drones lanzan proyectiles contra él. Solo desaparecen al recibir una cantidad determinada de daño.

Además de estos ataques, el Nerd es capaz de generar enemigos normales de ataque para que le ayuden en el combate. Esto sucede cuándo el Nerd recibe una cantidad determinada de daño. Mientras los enemigos generados no sean derrotados, el Nerd goza de un lapso de invulnerabilidad.

3.5.2.2.3. Jefe del tercer nivel

El jefe del tercer nivel en realidad son dos enemigos: la Madre y el Hijo.

Sus ataques se basan en lanzar proyectiles (pelotas) que, de forma similar a los drones especiales, rebotan en los obstáculos. Estos proyectiles desaparecen al dañar al jugador o tras sobrepasar un límite de tiempo.

La Madre se va moviendo entre puntos del escenario (puntos de patrulla) mientras lanza tres proyectiles contra el jugador de forma constante. Cuando alcanza un límite de lanzamientos, se coloca en el centro y realiza una serie de lanzamientos especiales en los que se dirigen proyectiles en todas direcciones, mientras el Hijo gira alrededor del centro redirigiendo los proyectiles alcanzados contra el jugador. Finalmente, tras realizar un número determinado de estos lanzamientos, la Madre vuelve a moverse entre los puntos del escenario.

El Hijo, a diferencia de la Madre, persigue en todo momento al jugador, excepto cuando la Madre realiza sus lanzamientos especiales. Al igual que la Madre lanza de forma constante tres proyectiles contra el jugador. Cuando el Hijo recibe daño con mucha frecuencia, la Madre se acerca a él e inician el llamado estado protección, en el que se mueven juntos durante un tiempo

determinado. Durante el estado protección solo ataca el Hijo y todo el daño que reciba es absorbido por la Madre.

Al morir uno de los dos, el otro entra en el estado furia, consiguiendo una bonificación de ataque, de velocidad de movimiento y que sus proyectiles sean más veloces y puedan rebotar más tiempo antes de desaparecer. En el caso de la Madre, en el estado furia comienza a perseguir al jugador y ya no necesita colocarse en el centro para realizar sus lanzamientos especiales. En cuanto al Hijo, este es capaz de redirigir contra el jugador cualquier proyectil (lanzado por él mismo) que lo alcance.

3.5.2.3. Componentes

Para implementar los nuevos enemigos, modificar los antiguos y conseguirse los nuevos comportamientos y mecánicas habrá que implementar nuevos componentes.

Por una parte, será necesario un componente relacionado con la IA del enemigo. Dicho componente contará con una Blackboard y un BT. El BT modelará el comportamiento del enemigo con una estructura en forma de árbol (tal y como hemos visto en el Estado del arte). La Blackboard servirá como un contenedor con las variables necesarias para el funcionamiento del enemigo, las cuáles serán consultadas y modificadas por el BT.

Por otra parte, también será necesario añadir componentes para regular las interacciones con el entorno, como, por ejemplo, los ataques. Dichos componentes serán administrados por el componente IA siendo accedidos, bien para consultar información o para realizar alguna determinada acción.

También será necesario algún componente controlador que inicialice el componente de IA, lo administre (que maneje de tiempos de carga de los ataques, por ejemplo) y que sirva de intermediario entre el componente de la IA y elementos externos. Por ejemplo, que sirva de intermediario cuando el jugador ataca al enemigo.

Será necesario componentes que representen rangos para la detección de elementos necesarios para la realización de acciones, como atacar al jugador, sanar a un enemigo, encontrar a un enemigo asustado, etc. En cuanto a los ataques, también se necesitará componentes centrados en ellos, que dañen directamente al jugador, generen y lancen proyectiles contra él, etc.

En el caso de los jefes, se necesitan además algunos componentes especiales. Para el Nerd, es necesario tener un componente que administre el movimiento del enemigo que lleve un control sobre los puntos clave y puntos intermedios del escenario, para luego seleccionar de forma semi-aleatoria un camino a través de los puntos intermedios hasta llegar a un punto clave. El Nerd necesita además componentes externos para sus ataques, como generadores externos de drones para el ataque especial y Spawners para generar a sus aliados.

También, el Nerd precisa que la estancia en la que se genera cuente con puntos clave y puntos intermedios, ya que sin ellos no podrá moverse. Para detectar la llegada a dichos puntos, precisará de algún componente de detección.

Lo mismo sucede con la Madre e Hijo, quienes precisan puntos semejantes al del Nerd para moverse y algún componente de detección para saber cuándo se ha alcanzado uno de esos puntos. En el caso del Hijo, solo precisará dichos puntos para moverse alrededor de la Madre.

3.5.3. Control de enemigos

3.5.3.1. Generación de los jefes

Para poder generar a los distintos jefes de nivel se añadirán nuevos Spawners (generadores de enemigos) en la sala del jefe, uno por cada jefe. Debido a que algunos necesitarán contar con cierta información sobre el escenario para poder funcionar correctamente, sus Spawners contarán con un componente responsable de configurar al jefe con los datos apropiados antes de generarlo.

Al contar con varios jefes, la solución original deja de ser válida y es necesario distinguir entre el caso de generar enemigos normales (enemigos que no son jefes) y generar un jefe. En el caso de los enemigos normales, el controlador de la sala (HallController) deberá activar todos los Spawners de la sala para generar la oleada de enemigos, mientras que en el caso de los jefes, el controlador deberá activar únicamente el Spawner o Spawners (recordemos que existe un jefe doble) relacionados con el jefe de dicho nivel y no activar todos los Spawners de la sala del jefe. Para ello, el controlador de la sala del jefe deberá consultar el número del nivel para conocer el jefe que debe generar.

3.5.3.2. Control de enemigos de apoyo

Como ya se ha dicho, los enemigos de apoyo se concentran detrás de los enemigos de ataque, es decir, en la retaguardia de los enemigos de ataque. Pero la retaguardia no es estática, durante el combate la retaguardia cambiará por culpa del movimiento del jugador y de los atacantes. Por ello es necesario calcular en todo momento la retaguardia de los enemigos de ataque.

El responsable de esta tarea será el controlador de la sala (HallController). El controlador llevará un registro de los enemigos de ataque y enemigos de apoyo vivos que se encuentren vivos en la sala.

Cuando haya enemigos de ataque y enemigos de apoyo en la sala, el controlador calculará la posición del grupo de atacantes a partir de la posición de cada enemigo de ataque. A partir de la posición del grupo de los enemigos de ataque y de la posición del jugador, el controlador calculará la posición del área de retaguardia y se la comunicará a todos los enemigos de apoyo que estén registrados. En el caso de que solo haya enemigos de ataque o enemigos de apoyo, el controlador dejará de calcular la retaguardia.

Cuando todos los enemigos atacantes de la sala mueren, el controlador de la sala avisa a los enemigos de apoyo que tenga registrados para que inicien su huida.

3.6. Plan de trabajo

Tras haber clasificado las distintas UT que componen el proyecto, podemos ver que aquellas que tienen un impacto directo en el producto pueden dividirse en dos grupos: aquellas referentes a los Enemigos Sandbox (los Enemigos sin integrar aún en el juego, es decir, de prueba) y los Enemigos finales (los Enemigos ya integrados en el juego). Para el proyecto se ha decidido priorizar la implementación de los Enemigos Sandbox frente a los Enemigos finales, ya que estos últimos se derivan de los primeros. El proyecto se considerará terminando cuando todas las UT o, si no todas, al menos aquellas que consigan un producto funcional, sean completadas.

Se decidió dividir el desarrollo del producto en sprints de dos semanas, teniendo al final de cada sprint una reunión con el tutor para revisar el estado del proyecto. Para llevar un mayor control sobre la organización y el seguimiento del proyecto se decidió usar una aplicación web en la que registrar las UT con sus PA y el estado de estas.

4. Diseño

4.1. Arquitectura del sistema

En esta sección de la memoria nos centraremos en la arquitectura de la aplicación.

En Black Friday War se pueden distinguir dos módulos principales, el módulo del nivel (Level Component) y el módulo de jugador (Player Component), los cuáles interactúan entre sí.

El módulo o componente del jugador cuenta a su vez con los subcomponentes de la familia, las habilidades y el equipamiento. El subcomponente de la familia es el responsable de administrar los atributos de los personajes del jugador como son el ataque, la velocidad, la vida o los PH (puntos de uso de habilidad), además de interactuar directamente con el módulo de nivel al desplazarse entre salas de nivel, luchar contra enemigos, etc. En cuanto al subcomponente de habilidades, este es el responsable de administrar las habilidades de cada personaje del jugador y su efecto sobre el jugador. Por último, encontramos el subcomponente del equipamiento, responsable de manejar los accesorios equipados por los personajes del jugador y su efecto sobre los atributos de este.

En cuanto al módulo del nivel, podemos distinguir tres subcomponentes: el nivel, las salas y los enemigos. El subcomponente del nivel es el responsable del cambio e inicialización de los niveles, además de seguir el progreso del jugador en cada nivel (es este subcomponente quién bloquea la sala del jefe hasta que no se hayan reunido los artículos clave). También es el subcomponente responsable de finalizar la partida cuando el jugador completa el último nivel o muere. En cuanto al subcomponente de las salas, dicho subcomponente es el responsable de administrar todo lo referente a las salas que componen el nivel, como es el ensamblaje (qué salas se conectan entre sí) hasta todo lo referente a lo que sucede en su interior (generación de enemigos, generación de ítems, bloqueo de puertas en combate…). Por último, encontramos el subcomponente de los enemigos, responsable del comportamiento de los enemigos que aparecen en el nivel.

Dicha arquitectura puede verse en la siguiente figura:

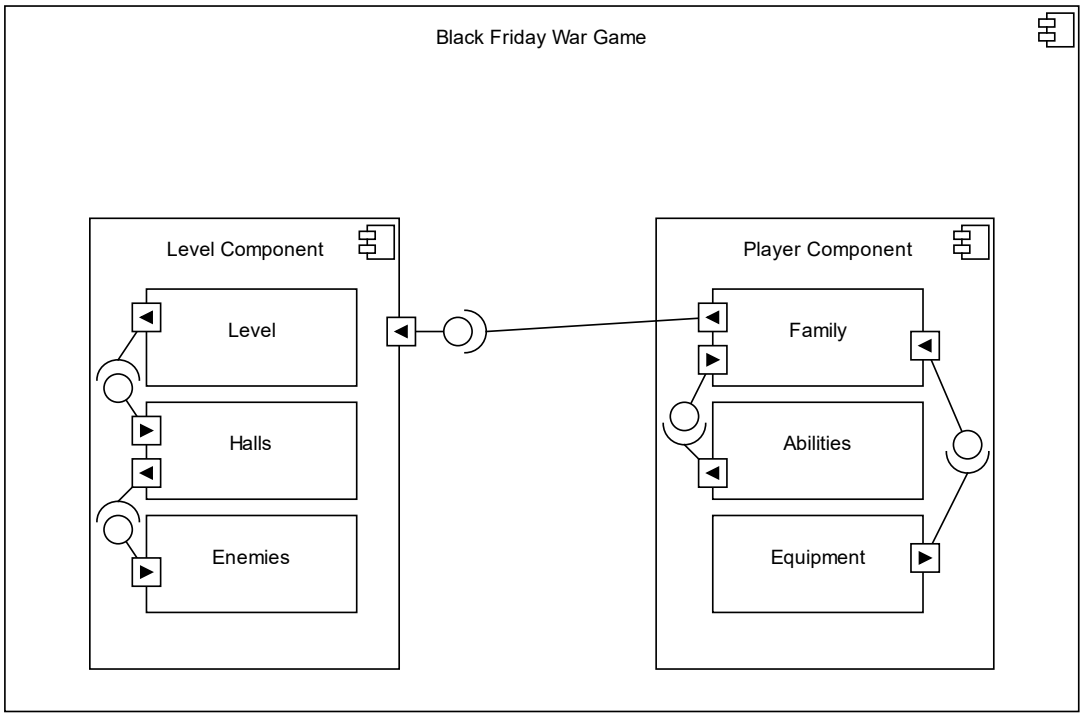


Figura 15: Arquitectura del sistema

En los siguientes apartados estudiaremos en más detalle el módulo del nivel, sobre todo lo referente a los enemigos.

4.2. Diseño detallado

4.2.1. Diseño de la IA de los Enemigos

Para implementar la IA de los Enemigos, como ya se mencionó anteriormente, se decidió usar la técnica de los Árboles de Comportamiento o Behavior Trees, técnica en la que el comportamiento de la entidad se vertebra en una estructura con forma de árbol.

A continuación, revisaremos el comportamiento de cada tipo de Enemigo.

4.2.1.1. Enemigos atacantes

Los enemigos atacantes, como dice el nombre, tienen como función la de atacar al jugador. Principalmente, un enemigo atacante persigue al jugador para reducir las distancias y poder atacarle.

Al recibir daño, su vida se reduce y sufre un pequeño “Stun”, una especie de retraso que le impide realizar acciones de forma temporal. Al ser dañados, si la vida que queda es menor a la mitad de la vida máxima, es posible que el enemigo se “asuste” en cuyo caso huirá del jugador mientras busca a un enemigo de apoyo. Si encuentra a un enemigo de apoyo, se dirigirá a él para que lo cure o lo anime (dependerá del tipo de apoyo) y no abandonará el estado asustado hasta que el enemigo de apoyo termine su función o hasta que el jugador les ataque. En el caso de que no encuentre un enemigo de apoyo, pasado un tiempo el enemigo volverá a la normalidad.

La curación del Enemigo Sanador hace que el enemigo recupere vida, mientras que la animación del Enemigo Animador aplica una mejora temporal en el ataque y la velocidad de movimiento del enemigo, además de evitar que vuelva a asustarse.

Cuando la vida del Enemigo es menor o igual a cero, este muere.

Este comportamiento se puede observar en la siguiente figura, que muestra el BT que siguen la mayoría de los Enemigos atacantes.

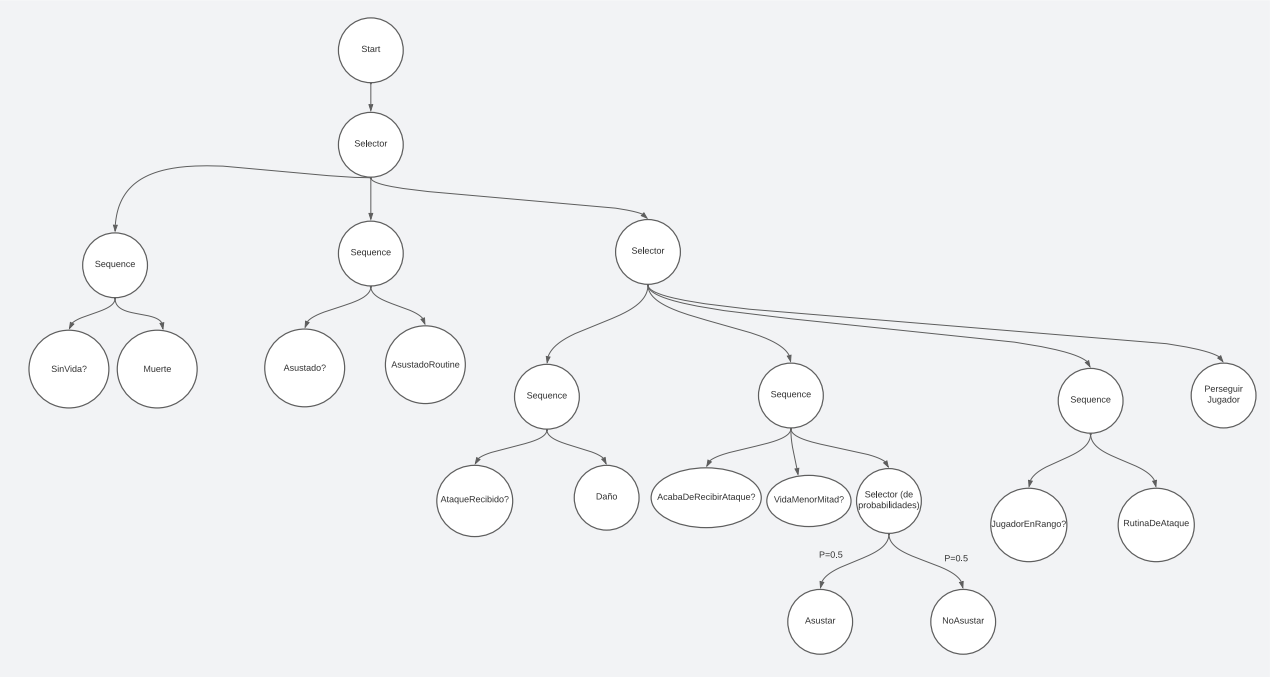


Figura 16: BT base de los enemigos atacantes

En cuanto a la rutina del Enemigo cuando se encuentra asustado (que en la figura anterior hemos llamado AsustadoRoutine) sería como la mostrada en la siguiente figura:

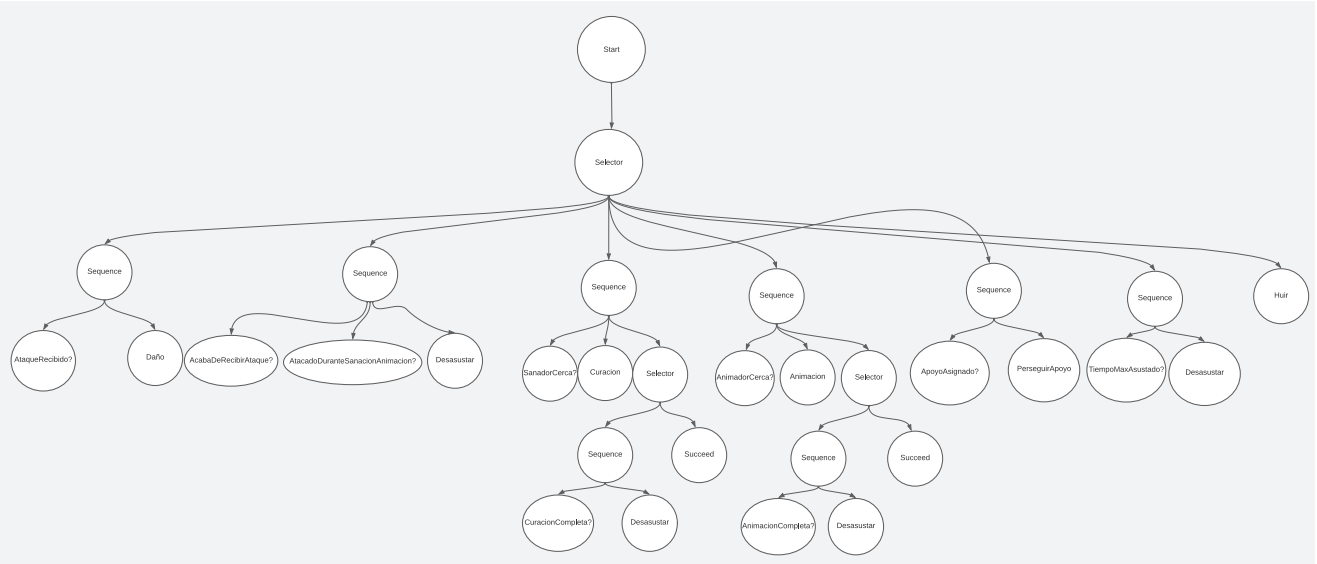


Figura 17: BT de la rutina de asustado

4.2.1.1.1. Enemigo Melee

El Enemigo Melee necesita estar a muy corta distancia del jugador para realizar su ataque, afectando al personaje del jugador más cercano al Enemigo.

La rutina de ataque del Enemigo a Melee se puede visualizar en la siguiente figura:

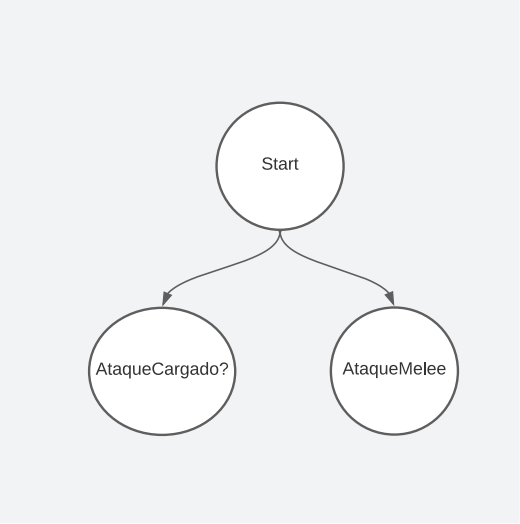


Figura 18: BT de la rutina de ataque del Enemigo Melee

4.2.1.1.2. Enemigo Distancia

El Enemigo Distancia no necesita tener muy cerca al jugador para atacarle, pero necesita que haya una trayectoria limpia entre el jugador y él, es decir, no debe haber obstáculos en medio. Cuando se cumple dicha condición, el Enemigo Distancia comienza a lanzar proyectiles contra el jugador.

La rutina de ataque del Enemigo Distancia se puede visualizar en la siguiente figura:

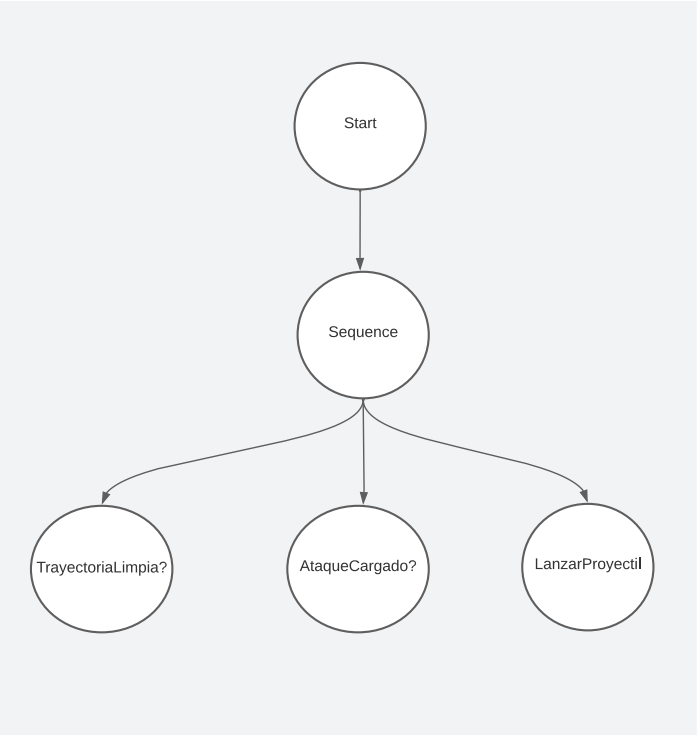


Figura 19: BT de la rutina de ataque del Enemigo Distancia

4.2.1.1.3. Enemigo Embestida

El Enemigo Embestida es un atacante con un comportamiento algo más complejo que el resto. Cuando dicho enemigo detecta al jugador dentro de su rango y existe una trayectoria limpia hasta el jugador (no hay obstáculos ni enemigos en ella), este, tras un período de preparación, inicia una carga en la que se lanza contra la posición en la que detectó al jugador.

Si durante la carga, el jugador es alcanzado, este es dañado y lanzado contra la pared en la dirección de la embestida, tras lo que el Enemigo inicia una retirada, en la que huye del jugador hasta que este le ataca o pasa un tiempo determinado, volviendo al comportamiento de siempre.

En el caso de que la embestida falle, el Enemigo no parará hasta golpear un obstáculo, tras lo que se quedará aturdido, quedándose inmóvil e incapaz de realizar cualquier acción hasta que no le ataque el jugador o pase un tiempo determinado, momento en el que regresará a su comportamiento normal. También es posible que alcance a un enemigo, en cuyo caso dañará a dicho enemigo e iniciará su retirada de la misma forma que lo habría hecho si hubiera embestido al jugador.

Su BT es un poco diferente al del resto de atacantes, siendo una especie de extensión del de los atacantes:

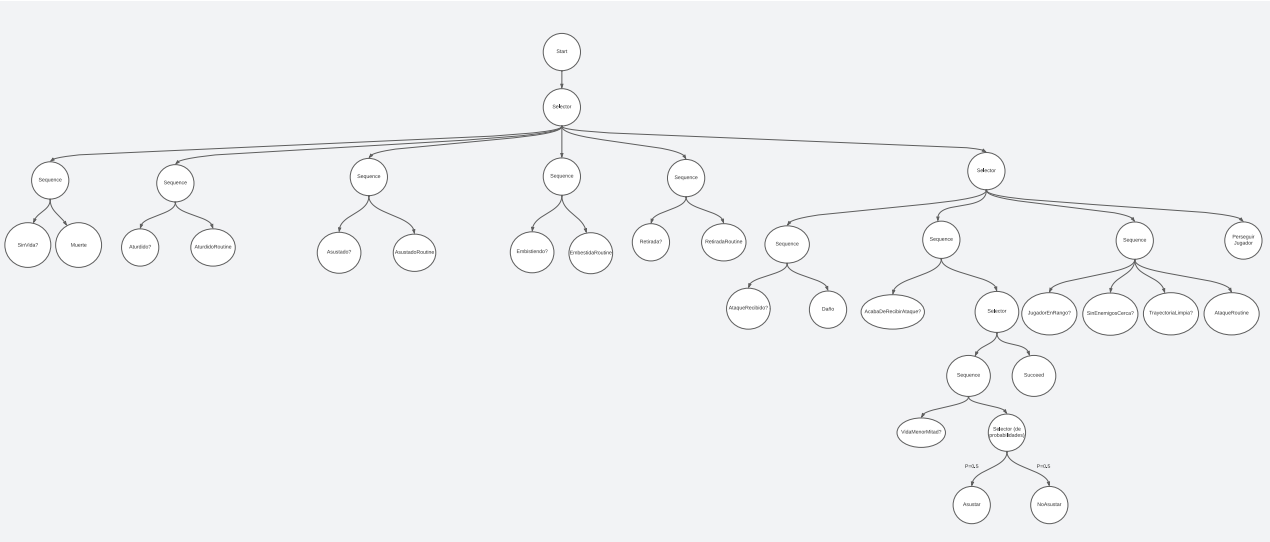


Figura 20: BT del Enemigo Embestida

Como se puede observar, su BT es similar al general de los atacantes, pero con nuevas rutinas. La rutina del aturdimiento sería la siguiente:

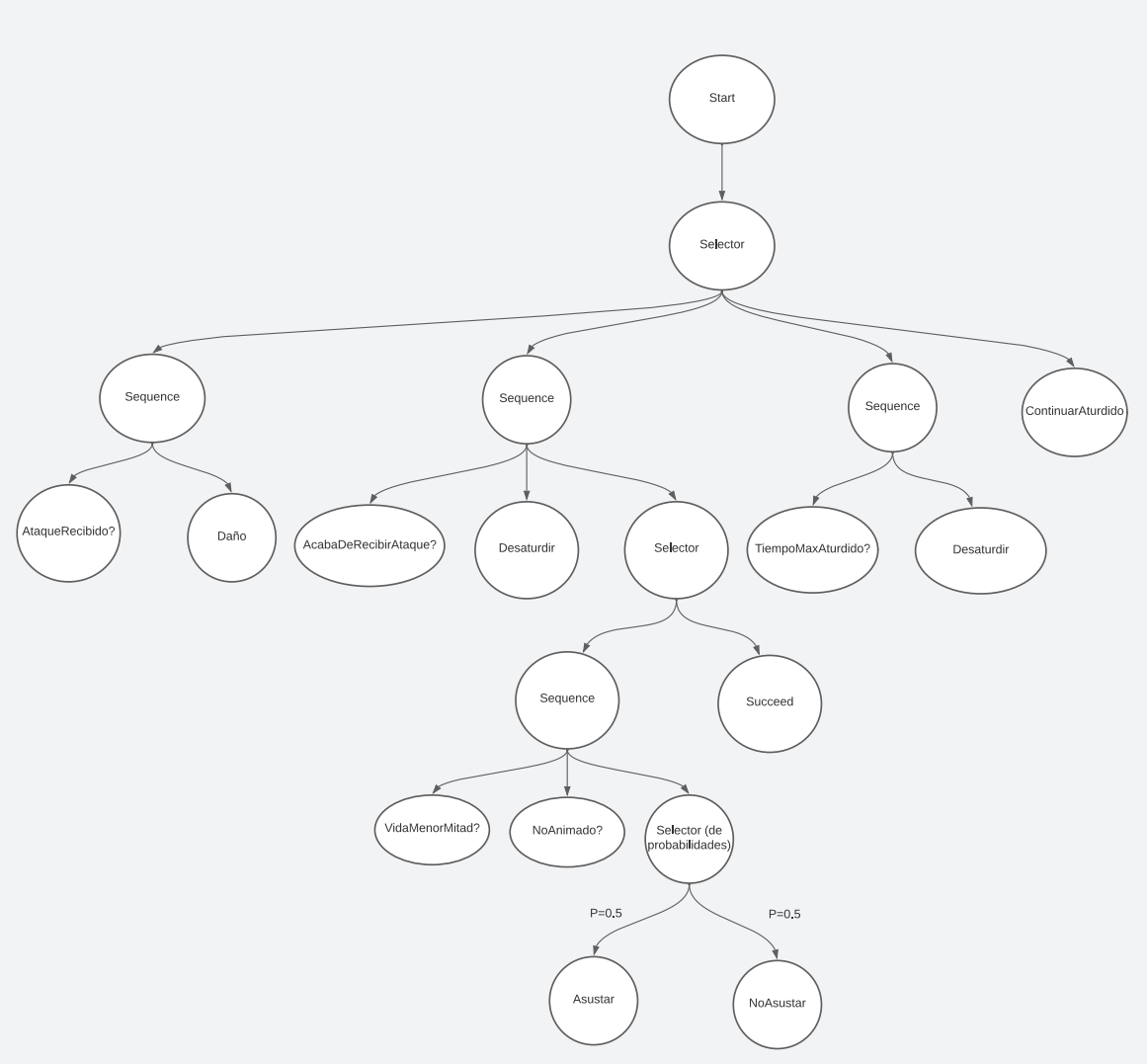


Figura 21: BT de la rutina de aturdimiento del Enemigo Embestida

En cuanto a la rutina de la embestida, esta tendría la siguiente forma en BT:

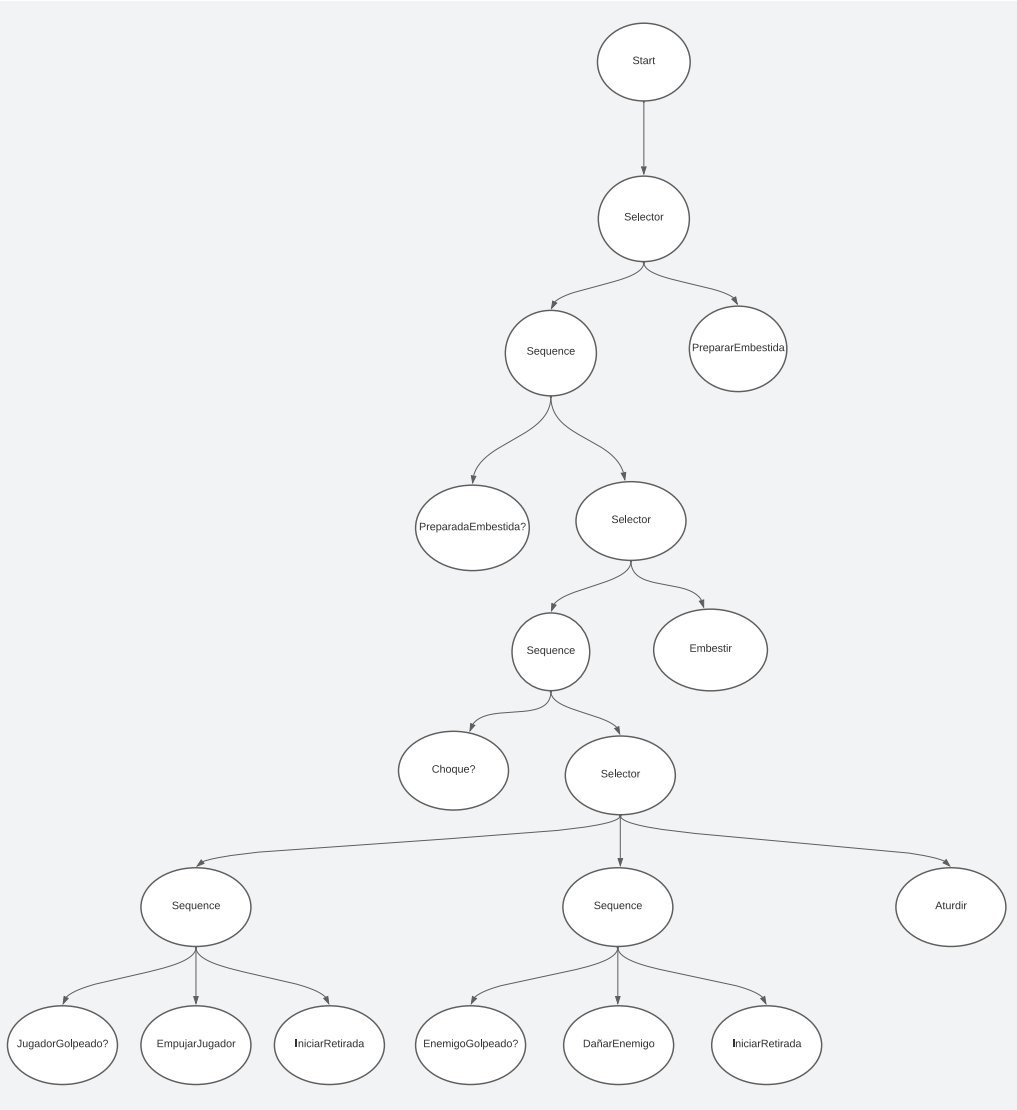


Figura 22: BT de la rutina de embestida del Enemigo Embestida

Por otra parte, la rutina de la retirada se vertebraría según el siguiente BT:

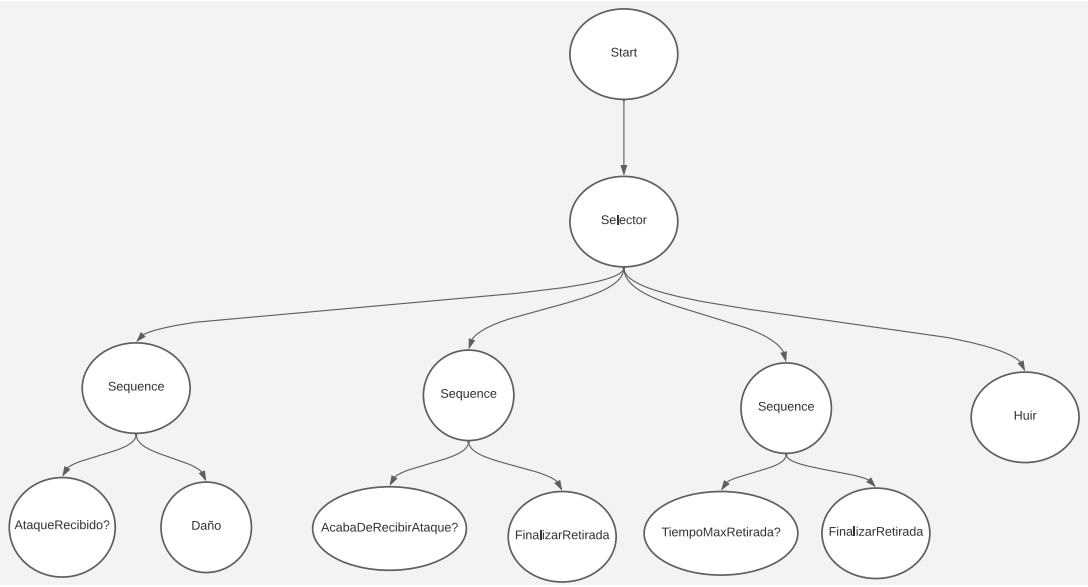


Figura 23: BT de la rutina de retirada del Enemigo Embestida

Por último, su rutina de ataque sería la siguiente:

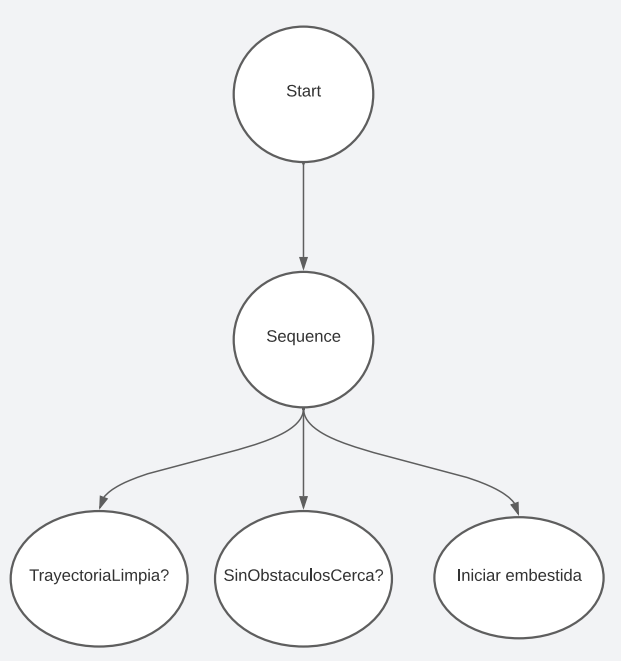


Figura 24: BT de la rutina de ataque del Enemigo Embestida

4.2.1.2. Enemigos de apoyo

La función de los enemigos de apoyo es la de ofrecer ayuda a los atacantes. Principalmente, un Enemigo de apoyo se centra en evitar al jugador mientras busca a enemigos asustados.

Cuando un enemigo de apoyo encuentra a un atacante asustado, este se dirige hacia él hasta estar lo suficientemente cerca para poder utilizar su “habilidad” de apoyo en él, tras lo que no abandonara al atacante hasta terminar el proceso o hasta que reciban un ataque del jugador.

Al igual que los enemigos atacantes, al recibir un ataque el enemigo de apoyo sufre un pequeño aturdimiento y ve reducida su vida. Cuando la vida es menor o igual a cero, el enemigo muere.

Originalmente se planteó la idea de que los enemigos de apoyo se movieran por el escenario a través de puntos establecidos, realizando así una especie de patrulla, hasta encontrar un enemigo asustado o encontrar al jugador. Si encontraba al jugador se alejaba hasta tener al jugador lo suficientemente lejos para proseguir con su patrulla.

Finalmente, esta idea para el movimiento de los enemigos de apoyo se descartó, ya que se observó que la patrulla hacía difícil que los apoyo encontraran a un atacante asustado, además de sentirse el movimiento del apoyo antinatural y estático.

Por ello se decidió que los apoyo se moverían en formación con los atacantes, manteniéndose en la retaguardia de estos en todo momento hasta encontrar un enemigo asustado al que atender, volviendo a la retaguardia tras terminar con el enemigo asustado. De esta forma se facilita que los atacantes asustados al retroceder encuentren un enemigo de apoyo, además de conseguir que el movimiento del apoyo se sienta más natural y dinámico.

Cuando no quedan atacantes en el escenario, los apoyos simplemente huyen del jugador.

La rutina de movimiento en retaguardia del apoyo sería la siguiente:

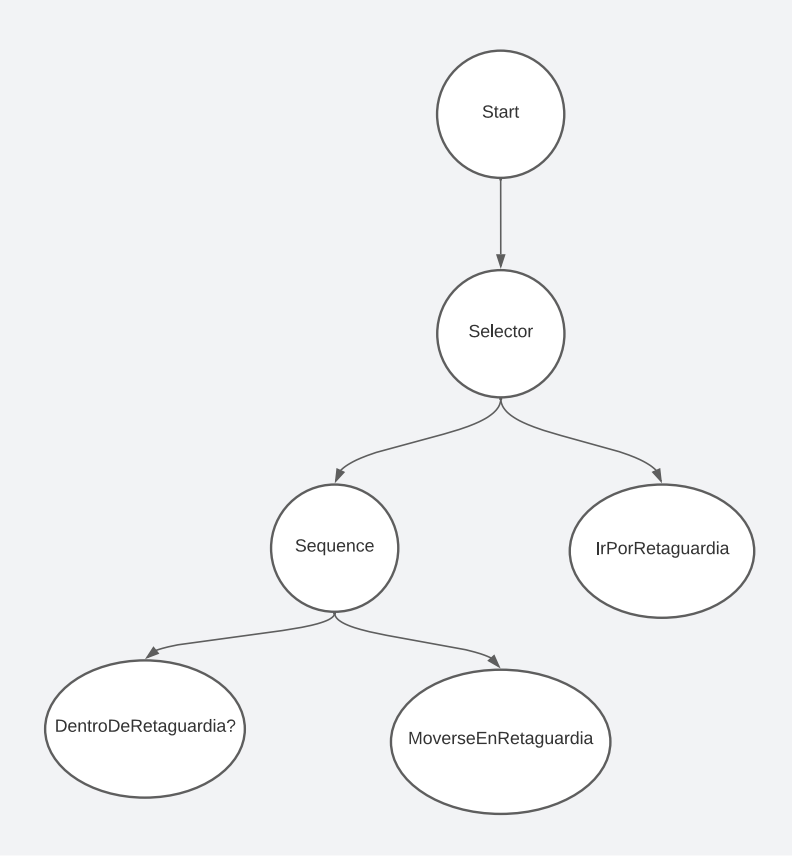


Figura 25: BT de la rutina de movimiento en retaguardia de los enemigos de apoyo

4.2.1.2.1. Enemigo Sanador

Como dice el nombre, la función del Enemigo Sanador es la de sanar enemigos atacantes. Cuando usa su habilidad de apoyo en un enemigo asustado, los puntos de vida de dicho Enemigo se van restaurando poco a poco a un ritmo constante, considerándose la sanación completada cuando su vida se halla rellenado del todo.

Su BT puede visualizarse en la siguiente figura:

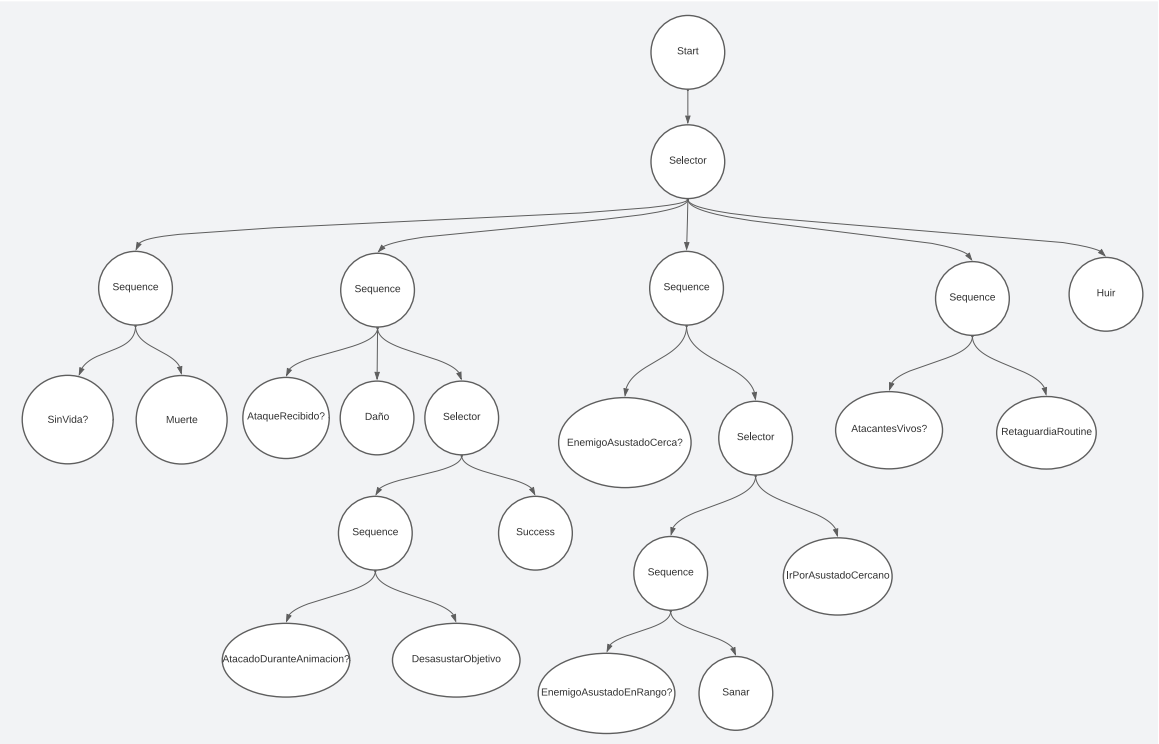


Figura 26: BT del Enemigo Sanador

4.2.1.2.2. Enemigo Animador

La función del Enemigo Animador es la de potenciar Enemigos atacantes. Cuando usa su "habilidad" de apoyo en un Enemigo asustado, este no recibe la potenciación hasta que no pasa un tiempo determinado, tras lo cual se considera que el proceso finalizó con éxito. Dicho proceso se denomina animación.

Su BT puede visualizarse en la siguiente figura, siendo bastante semejante al BT del sanador:

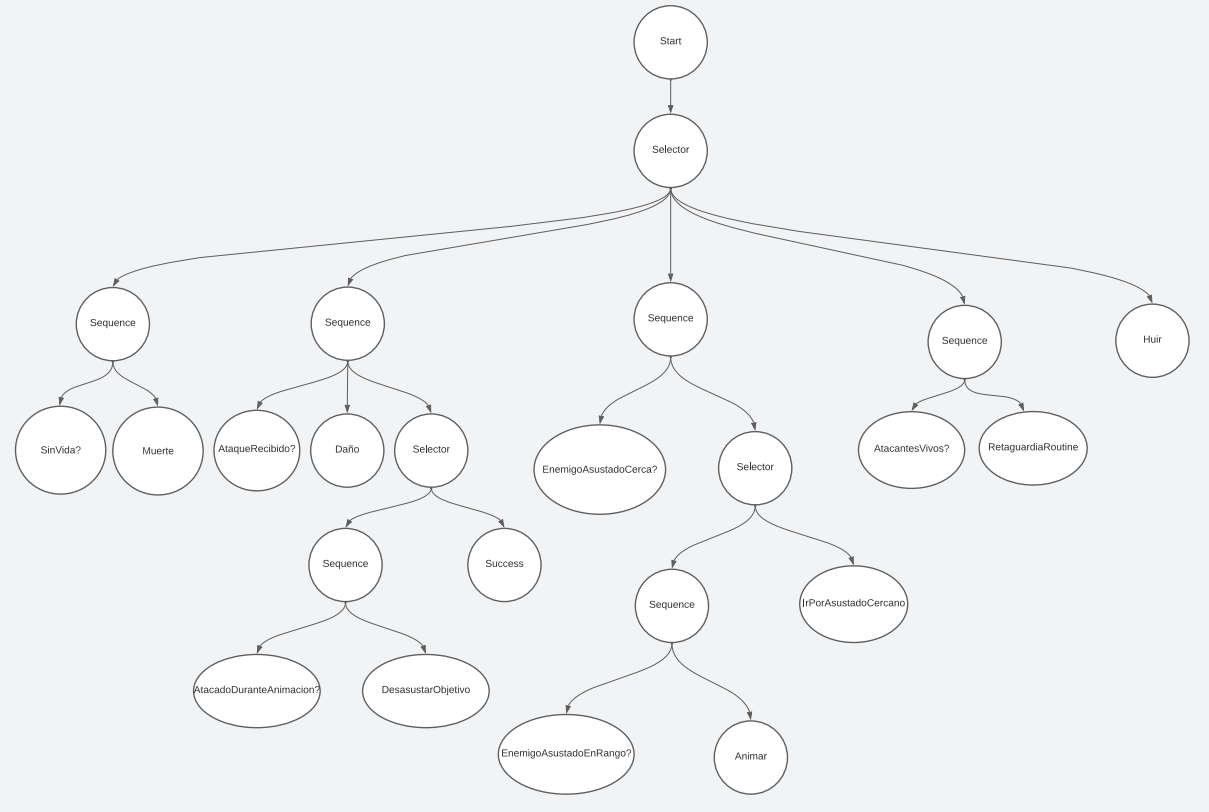


Figura 27: BT del Enemigo Animador

4.2.1.3. Jefes de nivel

Entre los jefes de nivel es difícil encontrar rutinas de comportamiento comunes entre ellos, ya que sus patrones de comportamiento difieren mucho de un jefe a otro. Sí que podemos destacar que, a diferencia de los Enemigo normales (Enemigos atacantes y de apoyo), al recibir un ataque, aunque su vida se ve reducida, no sufren ningún tipo de aturdimiento. También ha de mencionarse que, al alcanzar una cierta condición, normalmente referente a la vida del jefe, el patrón de ataque se ve modificado al entrar en el estado "furia".

Por último, al igual que todos los enemigos, los jefes mueren cuando su vida es menor o igual a cero.

4.2.1.3.1. Abuela

La Abuela, jefe del primer nivel, siempre persigue al jugador hasta tenerlo lo suficientemente cerca para realizar su ataque a melee, un ataque cuerpo a cuerpo que daña al jugador y lo lanza contra la pared. Mientras persigue al jugador, la Abuela no cesa de lanzar un determinado número de ráfagas de tres proyectiles contra este hasta llegar al límite, momento en el que tiene que esperar un tiempo hasta volver a poder lanzar una nueva oleada de ráfagas. Cuando el jugador se encuentra lo suficientemente cerca de la Abuela, esta para el ataque a distancia y empieza a cargar el ataque a melee.

Cuando la vida de la Abuela es menor o igual a la mitad, esta entra en el estado furia, aumentando su velocidad de movimiento y la velocidad de sus proyectiles. Además, el ataque a distancia se ve modificado, duplicándose el número de ráfagas por oleada y alternando entre la ráfaga de tres proyectiles y una ráfaga de ocho proyectiles en todas direcciones. En el estado furia, si la Abuela alcanza al jugador, el ataque a distancia no se desactiva.

Este comportamiento se ve reflejado en el siguiente BT:

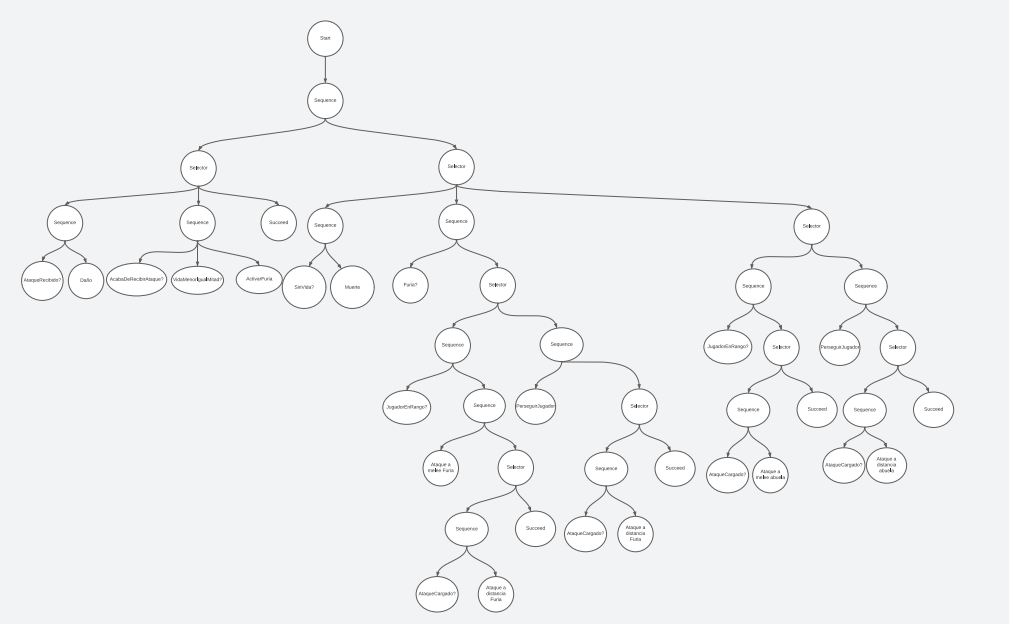


Figura 28: BT de la Abuela

4.2.1.3.2. Nerd

El Nerd, jefe del segundo nivel, no persigue al jugador. Este jefe se mueve de forma constante a través de puntos invisibles del escenario hasta llegar al extremo del escenario, en el que prepara un ataque especial en el que lanza una andanada de drones contra el jugador. A su vez, cada vez que atraviesa un punto invisible lanza un dron contra el jugador.

Tras el ataque especial, espera un tiempo hasta volver a moverse hacia otro extremo mientras lanza drones contra el jugador. El Nerd cuenta además con una habilidad que se activa cuando pierde una determinada cantidad de puntos de vida, con la que invoca Enemigos atacantes para defenderle y, mientras estos sigan vivos, los ataques del jugador no le afectan.

En cuanto al estado furia, al igual que la Abuela, este se inicia al ver su vida reducida a una cantidad menor o igual a la mitad, pero no sufre ninguna potenciación, sino que se añade una nueva habilidad que va generando constantemente, según una frecuencia dada, unos drones especiales que atacan al jugador.

Su BT sería el de la siguiente figura:

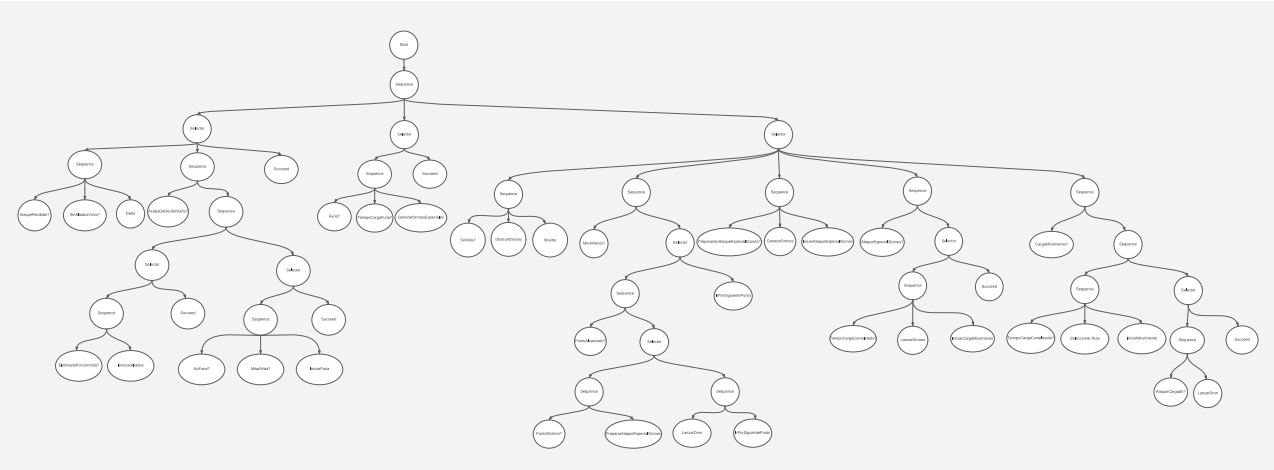


Figura 29: BT del Nerd

4.2.1.3.2.1. Drones del Nerd

Los drones del Nerd tienen un comportamiento algo más sofisticado que el de un proyectil normal, llegando a modelarse con un BT, por lo que se les puede considerar una especie de pseudo-enemigos.

4.2.1.3.2.1.1. Dron normal

El dron normal se dirige siempre hacia una determinada dirección hasta chocar con un obstáculo o con el jugador, momento en el que desaparece. Si choca con el jugador, además de desaparecer, el dron daña al jugador.

Al igual que los enemigos, el dron puede ser dañado por los ataques del jugador y, si se queda sin vida, desparece.

También hay que mencionar que el dron puede no tener activo el ataque, lo que hace que se quede inmóvil y no se mueva hacia el objetivo hasta que el ataque no esté activo. Esto se utiliza para el ataque especial del Dron, en el que, tras generarse los drones, el Nerd espera unos instantes antes de lanzarlos.

La BT del dron normal tendría la siguiente forma:

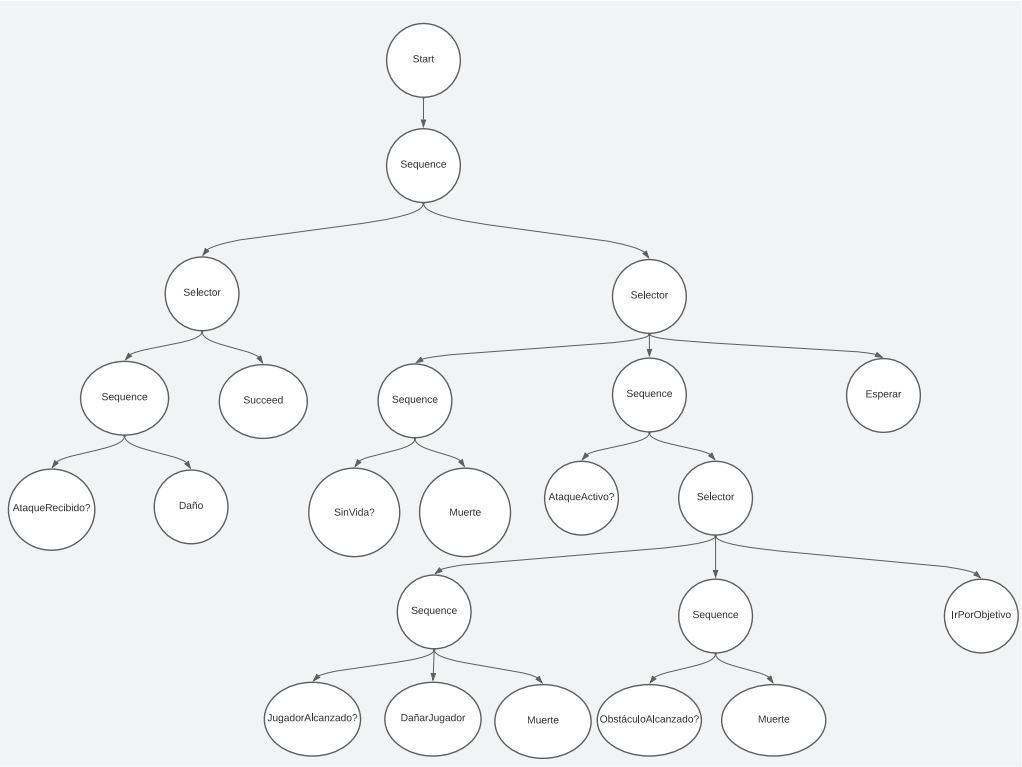


Figura 30: BT dron normal

4.2.1.3.2.1.2. Dron especial

El dron especial desplaza siempre hacia una dirección hasta que choca con un obstáculo o con el jugador, momento en el que cambia la dirección. Al chocar con el jugador, el dron también daña al jugador. Mientras se mueve, el dron no deja de lanzar proyectiles contra el jugador.

Al igual que el dron normal, este puede ser dañado por los ataques del jugador y, si se queda sin vida, desparece.

La BT del dron sería como sigue:

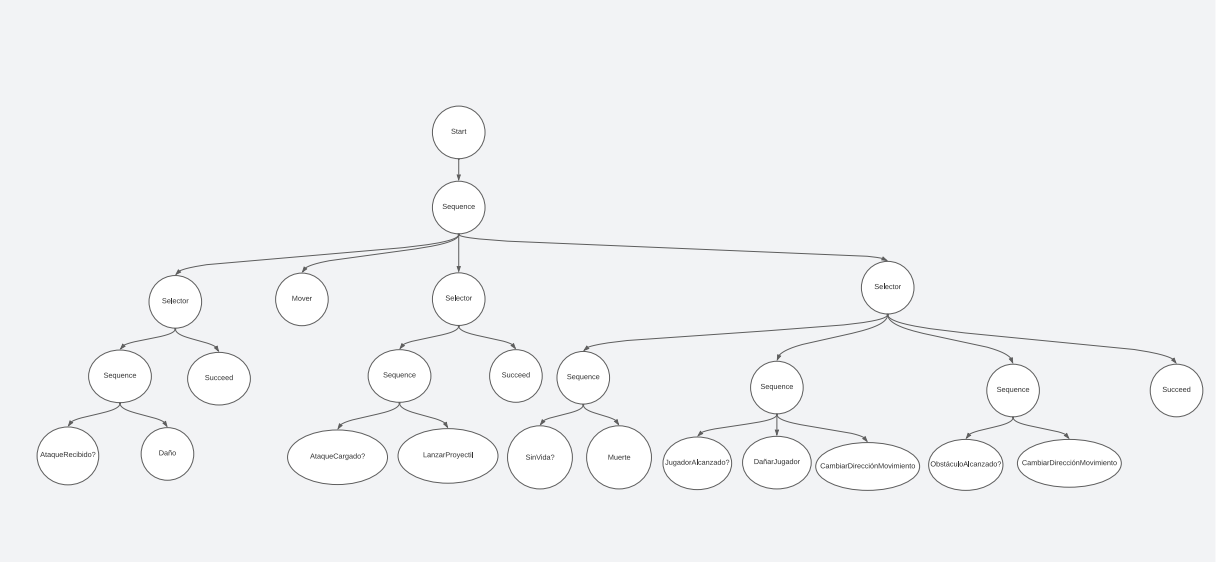


Figura 31: BT dron especial

4.2.1.3.3. Madre e Hijo

El jefe del tercer nivel destaca del resto al ser un "jefe doble", es decir, son dos enemigos: la Madre y el Hijo.

4.2.1.3.3.1. Madre Boss

Al principio, la Madre no persigue al jugador y se mueve de forma aleatoria entre puntos del escenario mientras lanza pelotas contra el jugador. Tras llegar a realizar un número determinado de lanzamientos de pelota, la Madre se coloca en el centro del escenario y, cuando el Hijo se coloca en un determinado punto por encima de ella, comienza una serie de lanzamientos especiales, en la que se lanzan pelotas en todas direcciones, hasta alcanzar un límite, momento en el que vuelve a moverse por el escenario mientras lanza pelotas contra el jugador.

Si el Hijo pide protección, la Madre deja de atacar y se dirige hacia él para ponerse junto a él e iniciar la llamada protección. Durante la protección, la Madre se mueve junto al Hijo, sin poder realizar ningún tipo de ataque mientras absorbe el daño recibido por el Hijo. Al pasar un tiempo, la Madre abandona la protección y vuelve a comenzar con sus ataques.

Al igual que el resto de los jefes de nivel, la Madre cuenta con un estado furia, pero, a diferencia del Nerd o la Abuela, el estado furia de la Madre no se activa al reducirse a la mitad su vida, sino que se activa al detectar la muerte del Hijo. Durante el estado furia, el ataque, la velocidad de movimiento de la Madre y la velocidad de las pelotas lanzadas aumentan. También comienza a perseguir al jugador. En cuanto a sus ataques, siguen alternándose los lanzamientos normales con los especiales al llegar al límite de cada uno, pero durante el estado furia la Madre es capaz de realizar los lanzamientos especiales sin colocarse en el centro del escenario ni esperar al Hijo (qué ya ha sido vencido).

Su BT sería el de la siguiente figura:

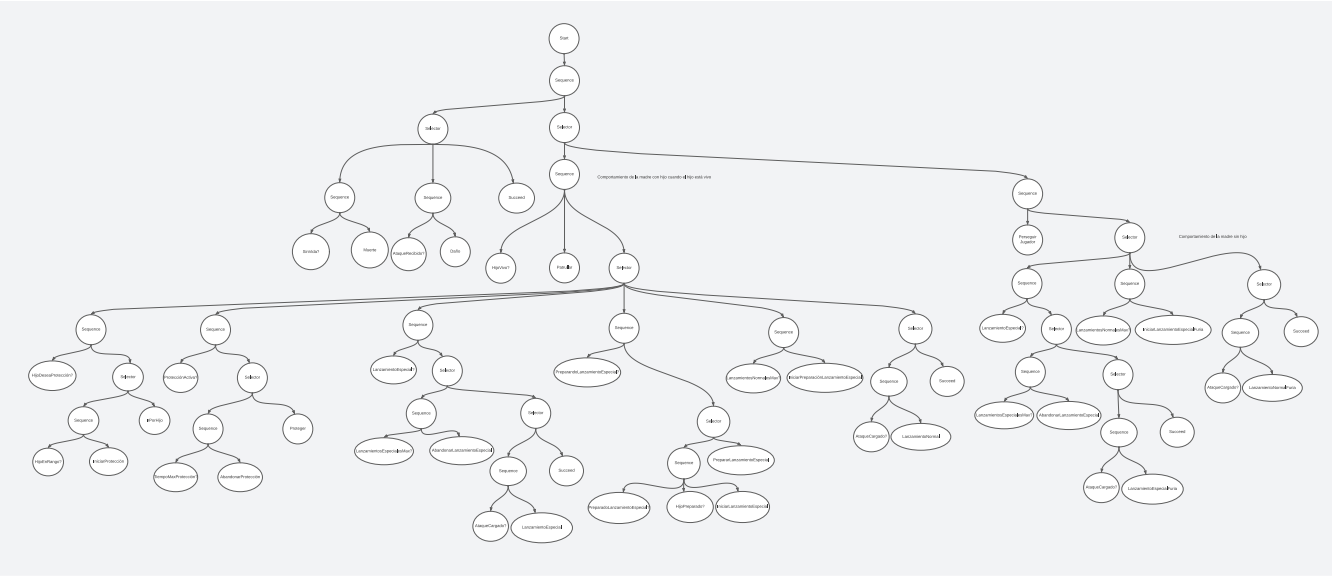


Figura 32: BT Madre

4.2.1.3.3.2. Hijo

A diferencia de la Madre, el Hijo persigue desde el principio al jugador, mientras lanza pelotas contra él. Cuando la Madre alcanza el límite de lanzamientos normales y comienza a preparar su andanada de lanzamientos especiales, el Nerd deja de atacar y se coloca en un determinado punto por encima del centro del escenario y, cuando la Madre inicia sus lanzamientos especiales, el Hijo comienza a moverse en círculos alrededor de la Madre, redirigiendo cualquier pelota que alcance contra el jugador. Cuando la Madre termina sus lanzamientos especiales, el Hijo vuelve a perseguir al jugador mientras lanza pelotas contra él.

Cuando el Hijo recibe muchos ataques con demasiada frecuencia, este pide la protección de la Madre y se acerca a ella para iniciar la llamada protección, durante la que el Hijo persigue al jugador y lanza pelotas contra él, siguiendo su rutina de ataque básica, pero su velocidad cambia por la de la Madre, quien se mueve junto al Hijo hasta que termine la protección. Durante la protección, todo el daño recibido por el Hijo es redirigido a la Madre, siendo el Hijo invulnerable durante la protección.

De forma similar a la Madre, el Hijo entra en el estado furia cuando la Madre es derrotada. Durante el estado furia, su ataque, su velocidad de movimiento y la velocidad de las pelotas lanzadas aumenta. Durante el estado furia, el Hijo sigue persiguiendo al jugador mientras le lanza pelotas, pudiendo además redirigir las pelotas que le alcanzan contra el jugador (al igual que durante el lanzamiento especial).

Su BT sería como el de la siguiente figura:

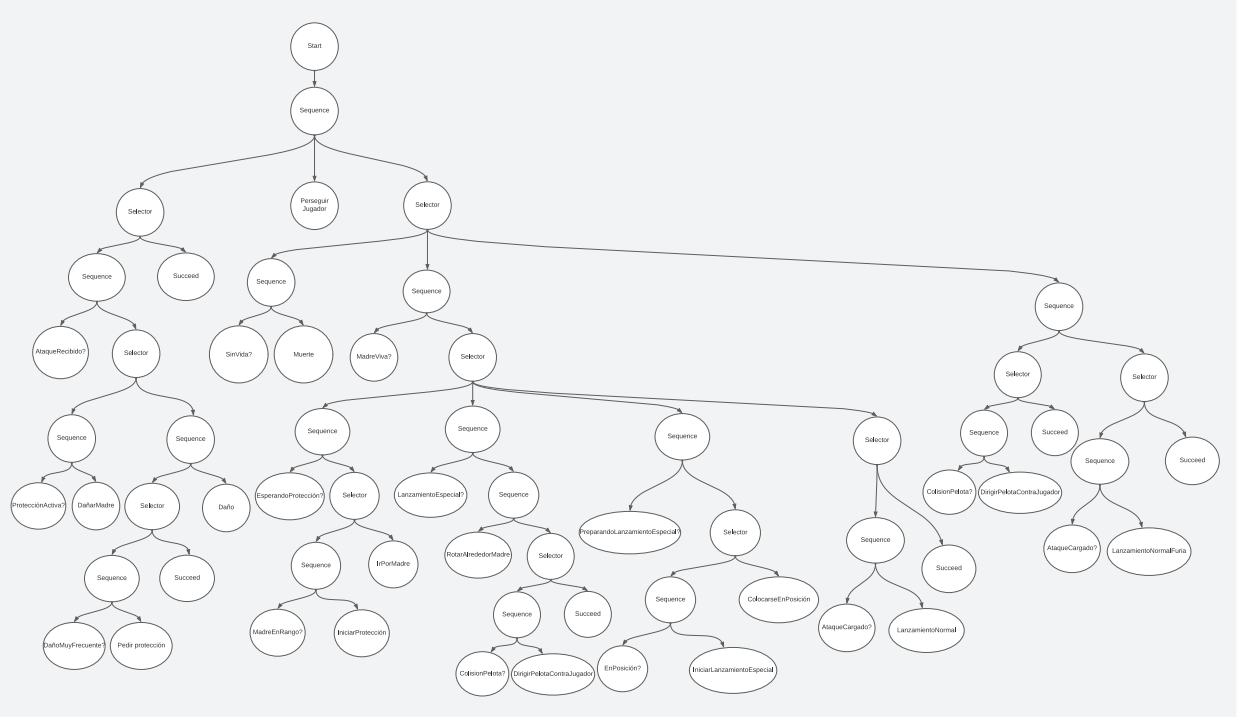


Figura 33: BT Hijo

4.2.1.3.3.3. Pelota

Al igual que los proyectiles lanzados por el Nerd (los drones), los proyectiles lanzados por la Madre e Hijo Boss tienen también un comportamiento complejo, por lo que se decidió modelarlo con un BT.

Dichos proyectiles, denominados pelotas, tienen un comportamiento similar al de los drones especiales del Nerd, cambiando la dirección de movimiento cada vez que chocan con un obstáculo (simulando así que rebotan). Las pelotas solo desaparecen cuando golpean al jugador o cuando se termina su tiempo de vida.

Su BT sería como la que sigue:

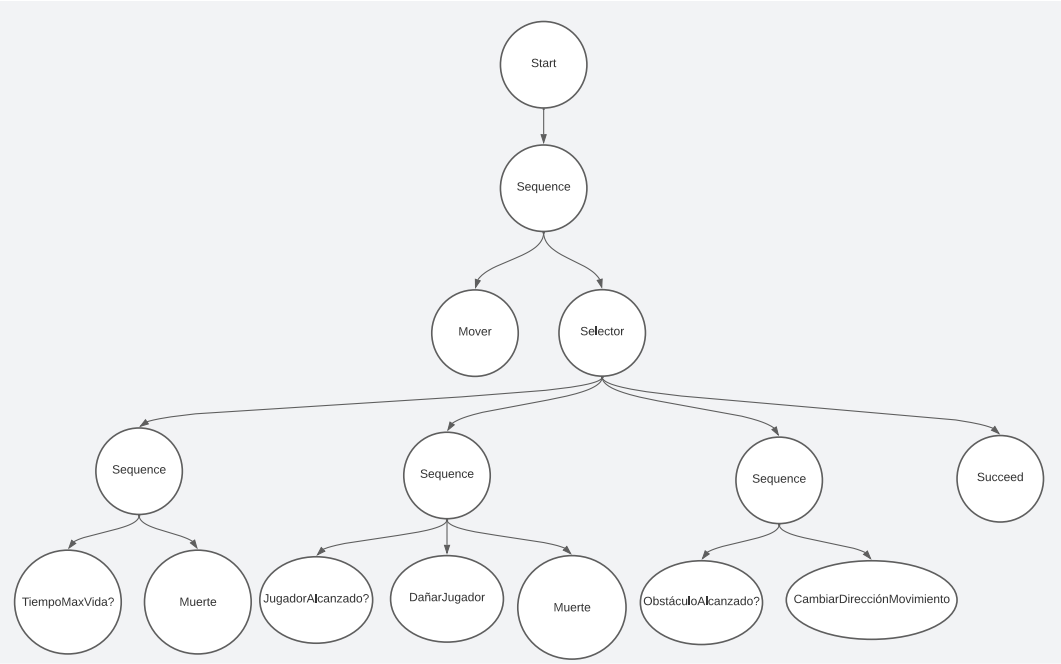


Figura 34: BT Pelota

4.2.2. Diseño de clases

A continuación, se detalla las clases del módulo del nivel, centrándonos en lo referente a los enemigos, por lo que no se llegarán a detallar componentes dentro del módulo que no tenga relación directa o indirecta con el comportamiento de los enemigos o su administración.

4.2.2.1. Nivel

En la siguiente imagen se puede ver el diagrama UML referente a los componentes responsables de la administración de los niveles.

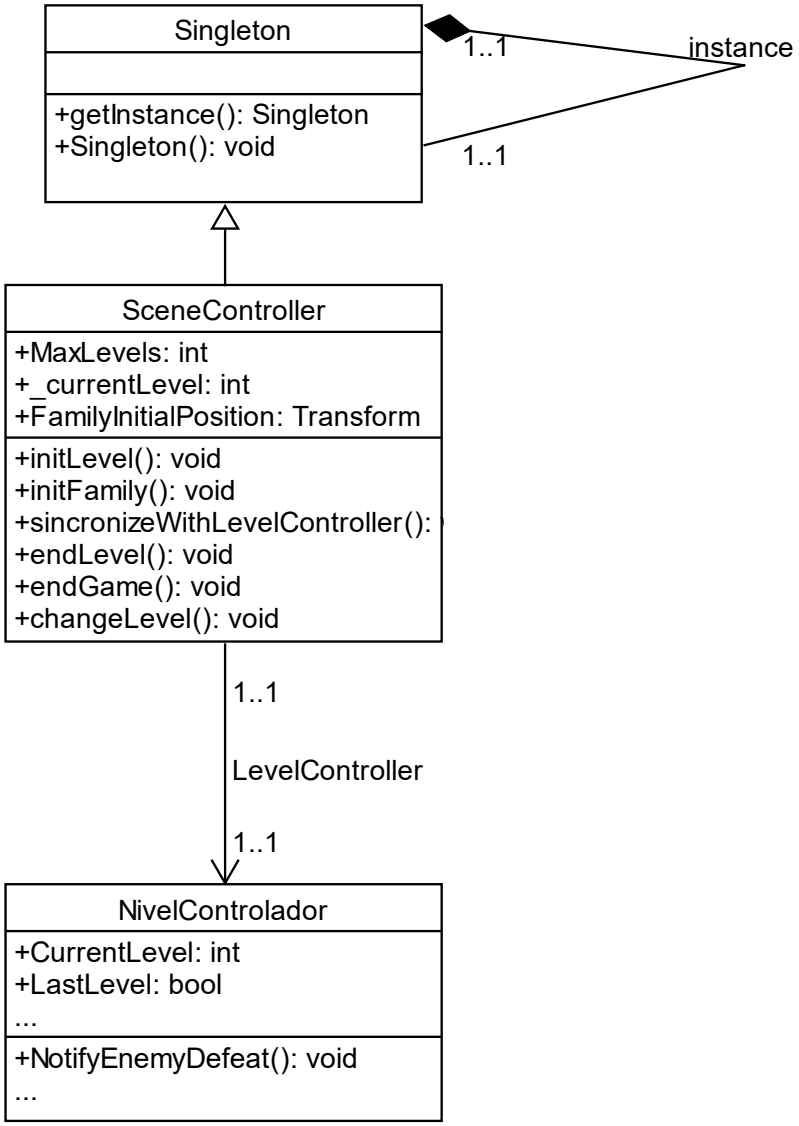


Figura 35: UML nivel

4.2.2.2. Salas y generación de enemigos

En la siguiente imagen se puede observar el diagrama UML referente a los componentes responsables de la administración de las salas y los enemigos del nivel.

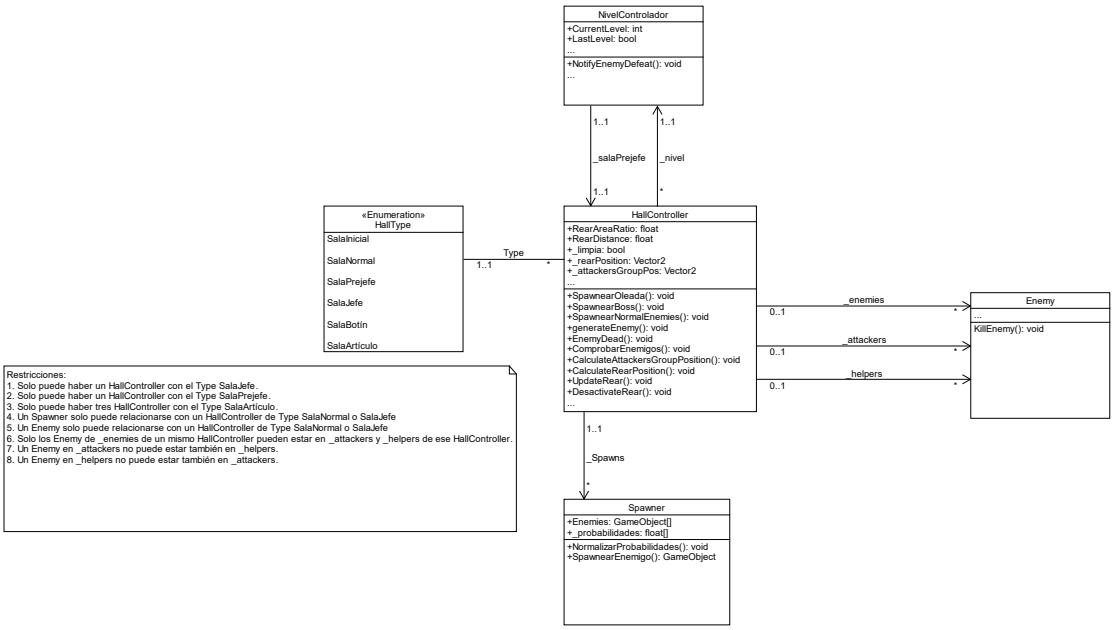


Figura 36: UML salas

4.2.2.3. Enemigos

Para los enemigos se decidió usar una clase controlador que administre el componente de IA del enemigo y que haga de intermediario en algunas interacciones. Como se puede observar en la siguiente figura, existen diversos controladores dependiendo del tipo de enemigo.

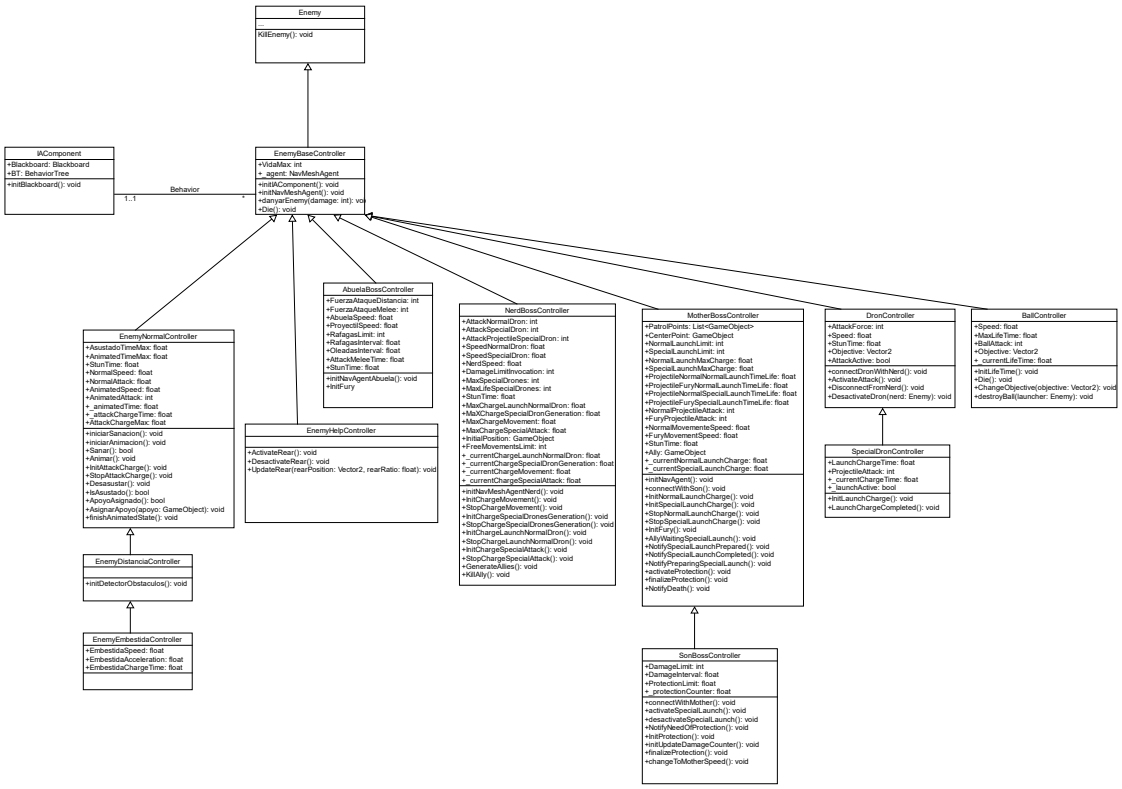


Figura 37: UML controladores de los enemigos

A continuación, revisaremos el diseño del IAComponent de cada enemigo. Para facilitar la comprensión de su estructura, se ha decidido representar a los BT como clases, uniendo los subárboles con el BT principal mediante una agregación. Las BT con el mismo nombre representan un mismo subárbol que se reutiliza para construir el BT de un nuevo enemigo.

En cuanto a los componentes conectados a la BT, como EnemyRange o MeleeAttack, aunque se hayan representado de forma separada de un diagrama a otro, son los mismos componentes reutilizados para otros enemigos. Se ha decidido realizar así la representación para facilitar la comprensión de cada estructura de forma independiente.

4.2.2.3.1. Enemigo Melee

En la siguiente figura se puede observar la estructura interna del IAComponent del Enemigo Melee.

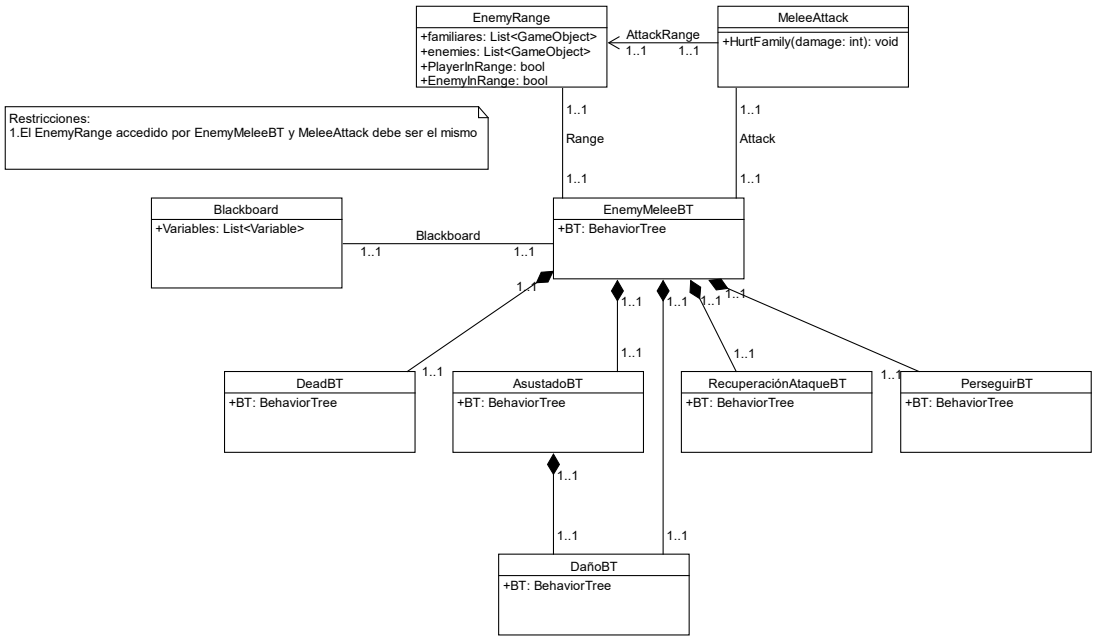


Figura 38: IAComponent del Enemigo Melee

4.2.2.3.2. Enemigo Distancia

En la siguiente figura se puede observar la estructura interna del IAComponent del Enemigo Distancia.

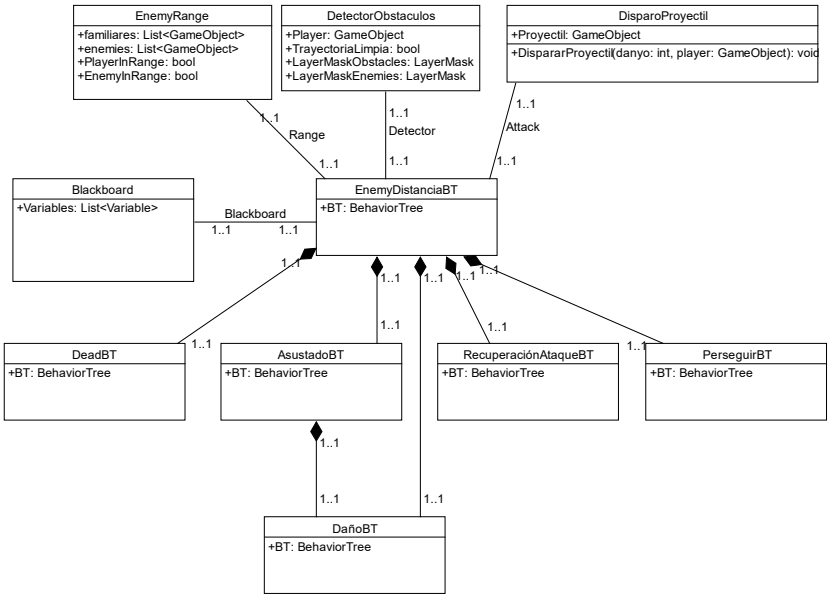


Figura 39: IAComponent del Enemigo Distancia

4.2.2.3.3. Enemigo Embestida

En la siguiente figura se puede observar la estructura interna del IAComponent del Enemigo Embestida.

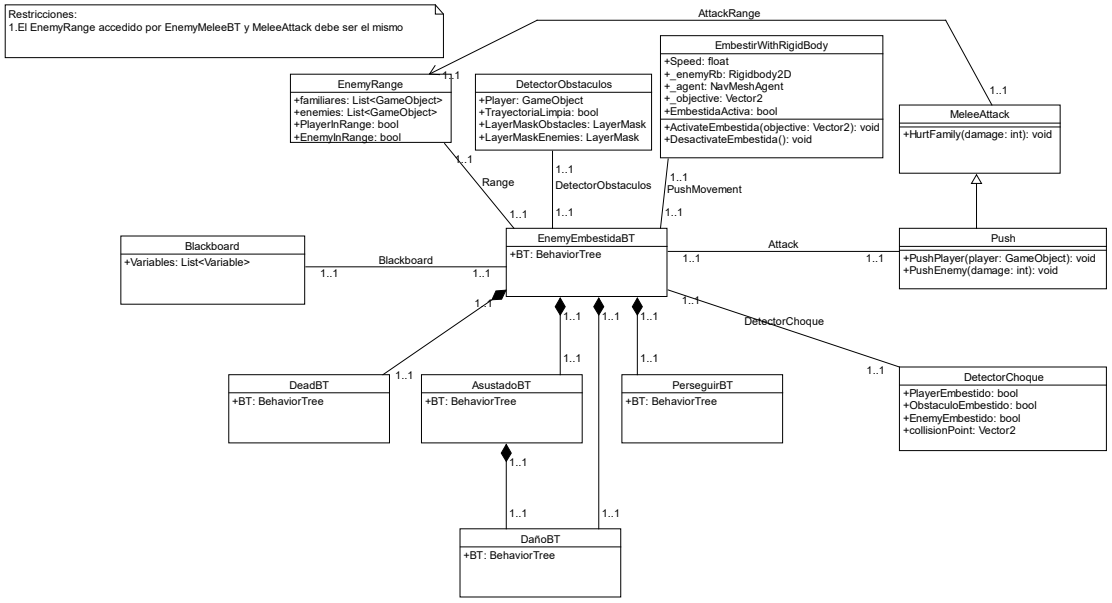


Figura 40: IAComponent del Enemigo Embestida

4.2.2.3.4. Enemigo de apoyo (Sanador y Animador)

En la siguiente figura se puede observar la estructura interna del IAComponent del Enemigo Sanador.

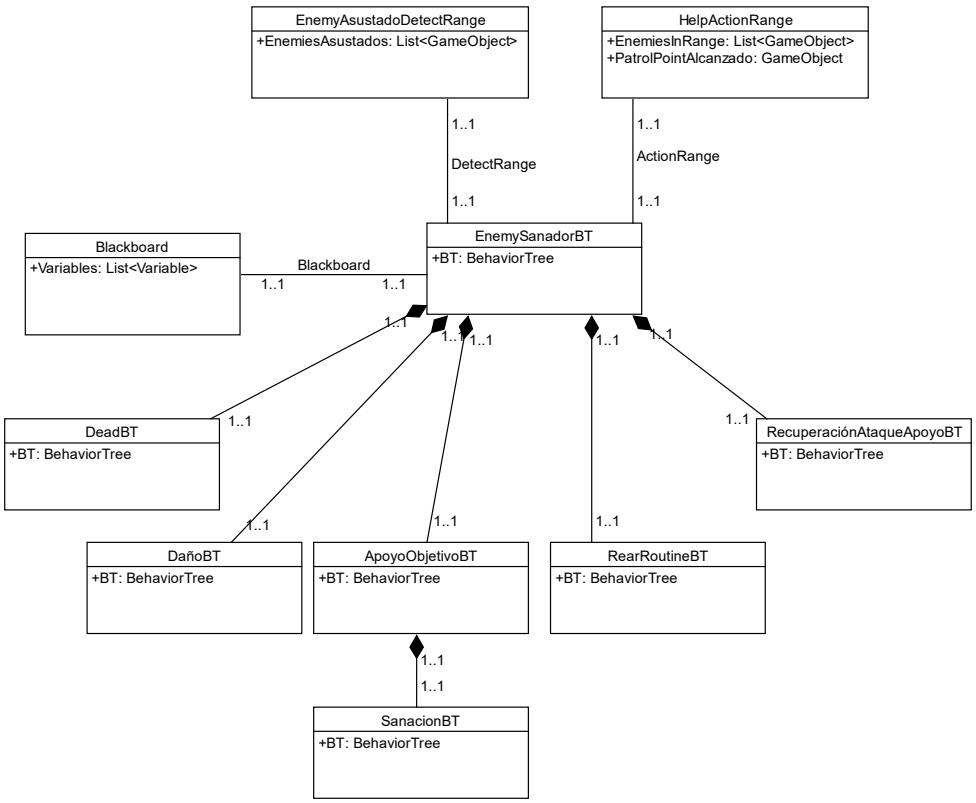


Figura 41: IAComponent del Enemigo Sanador

En cuanto al Enemigo Animador, su estructura es similar al del sanador.

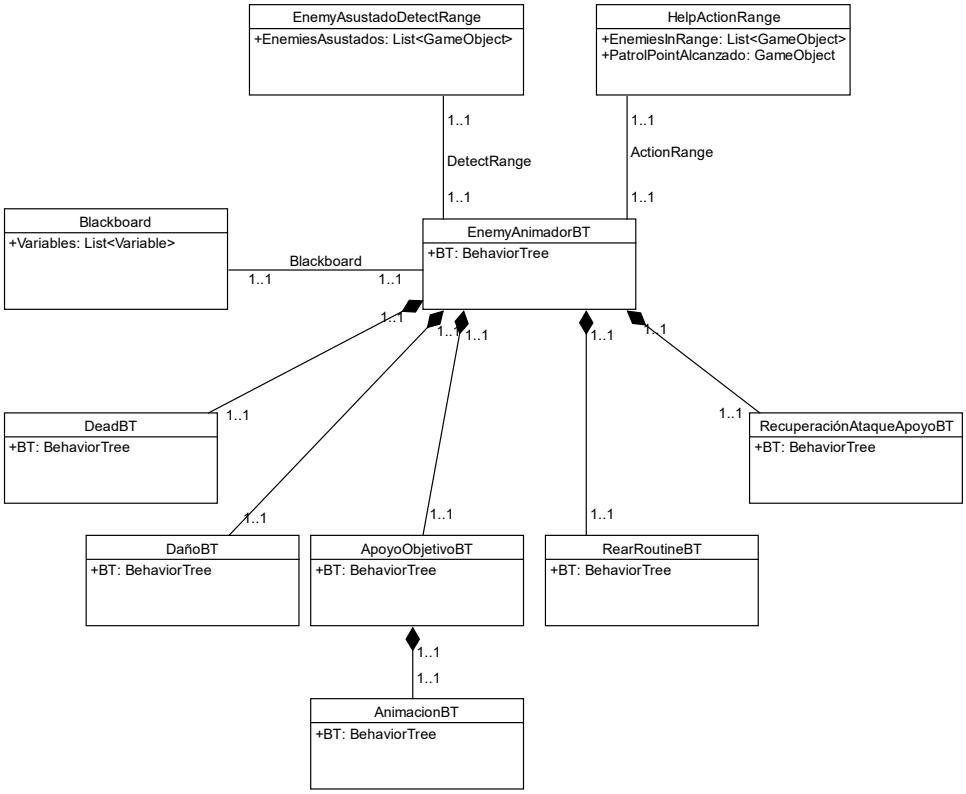


Figura 42: IAComponent del Enemigo Animador

4.2.2.3.5. Abuela

En la siguiente figura se puede observar la estructura interna del IAComponent del jefe del primer nivel: la Abuela.

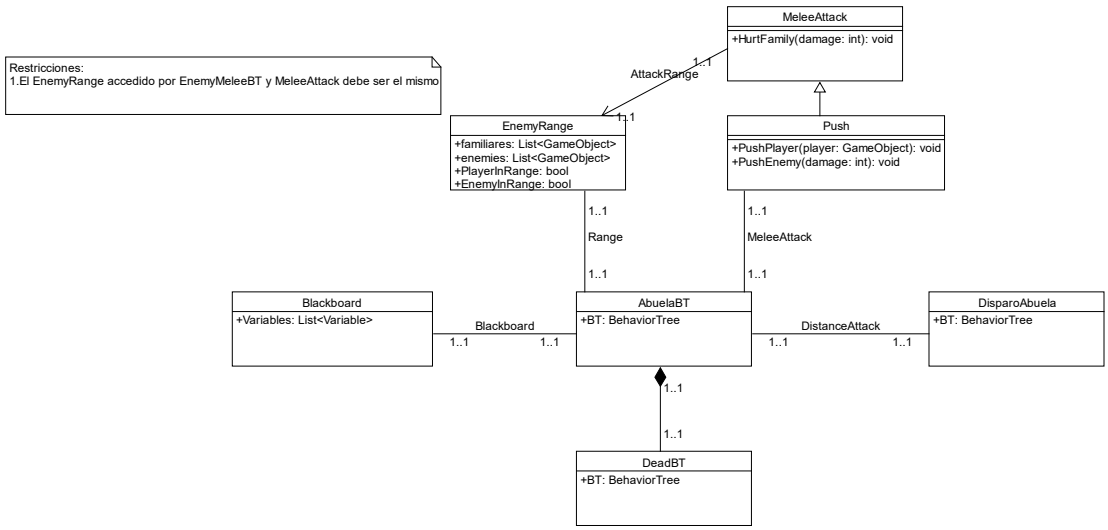


Figura 43: IAComponent de la Abuela

4.2.2.3.6. Nerd

En la siguiente figura se puede observar la estructura interna del IAComponent del jefe del segundo nivel: el Nerd.

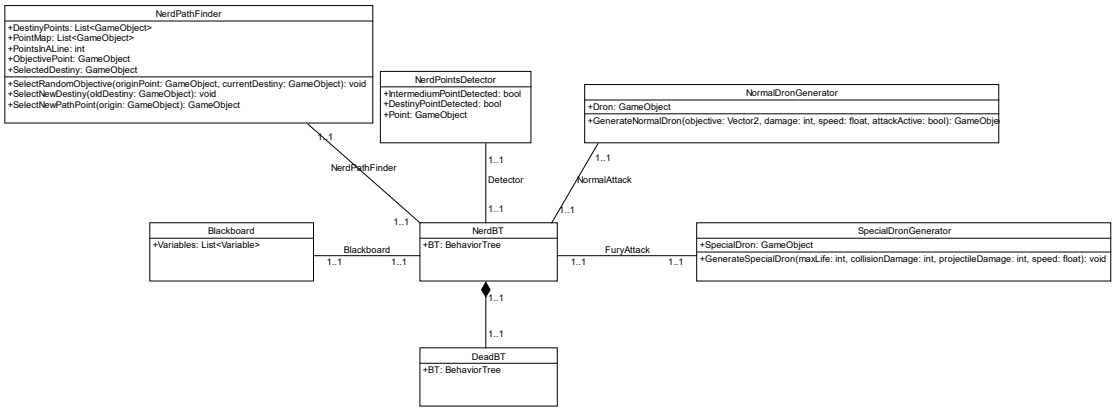


Figura 44: IAComponent del Nerd

En cuanto a sus drones normales y especiales, estos tienen las siguientes estructuras, que guardan ciertas similitudes entre ellas.

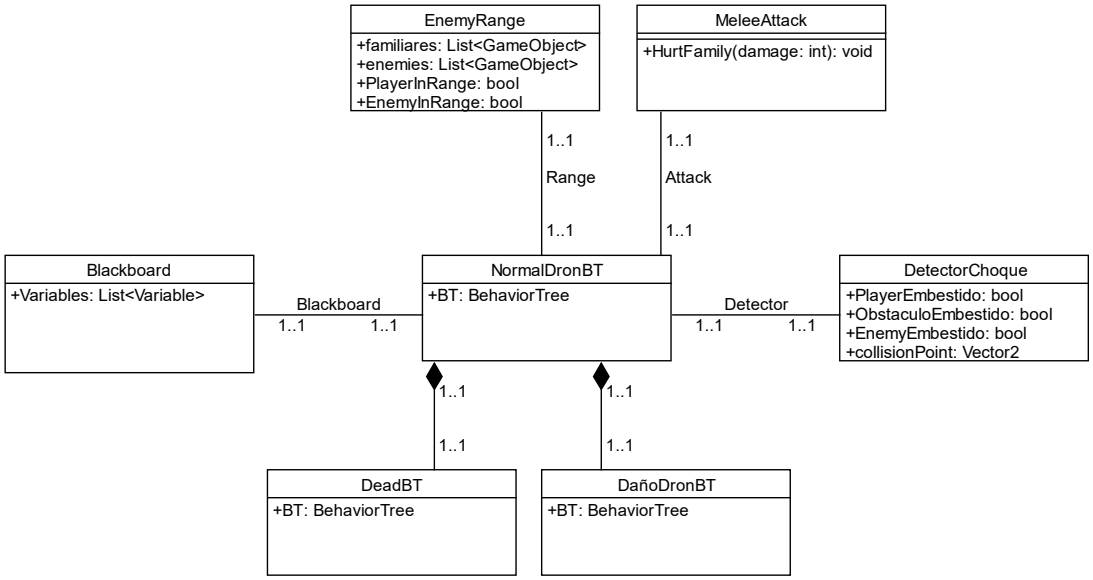


Figura 45: IAComponent del dron normal

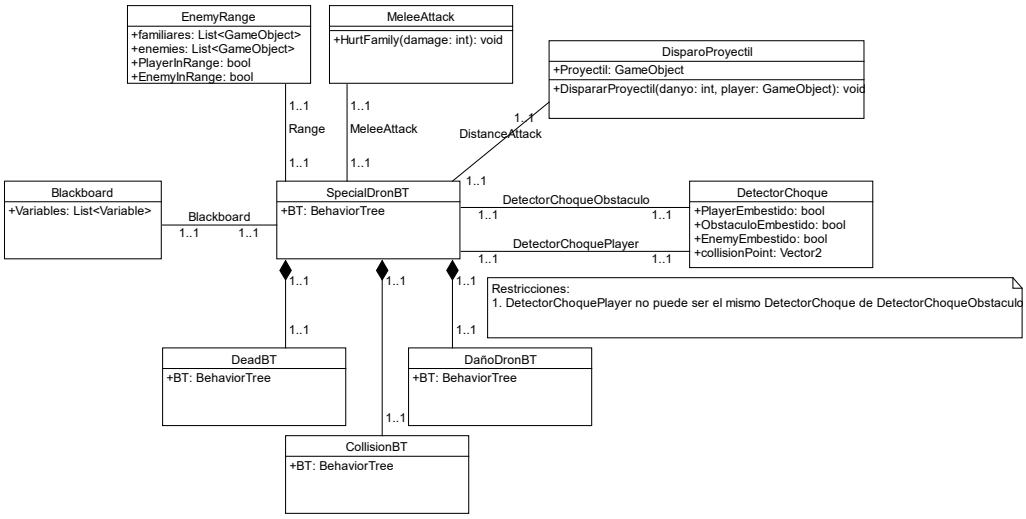


Figura 46: IAComponent del dron especial

Por último, es necesario mostrar la estructura del sistema de generación externo de drones, utilizado para crear la hilera de drones que se lanzan durante el ataque especial del Nerd.

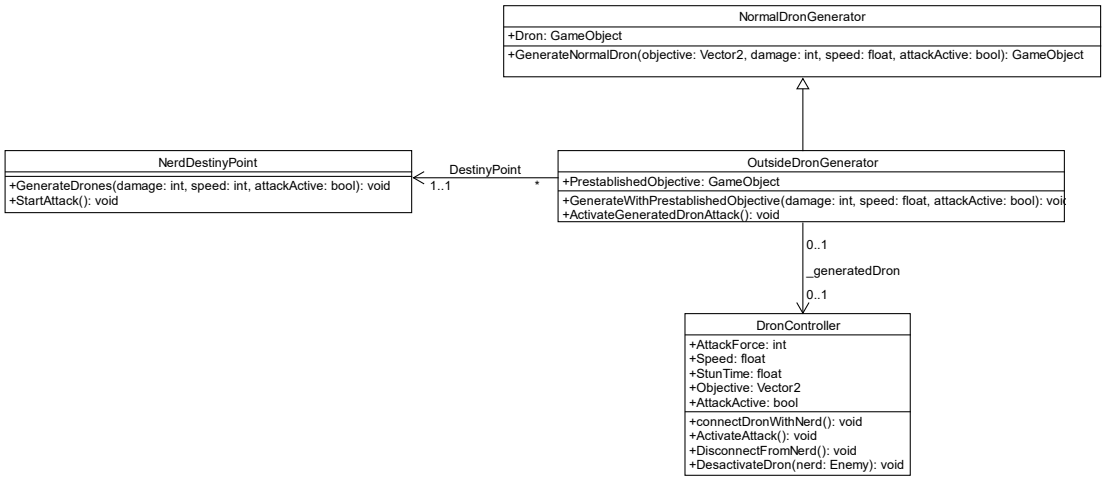


Figura 47: Sistema de generación de drones externo

4.2.2.3.7. Madre e Hijo

La estructura interna del IAComponent de los enemigos que componen el jefe del tercer nivel (un jefe doble) es la siguiente.

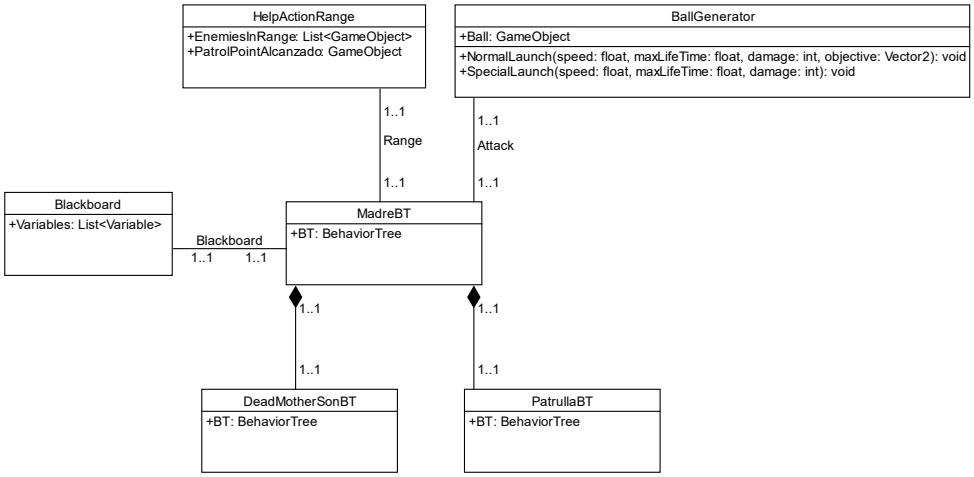


Figura 48: IAComponent de la Madre

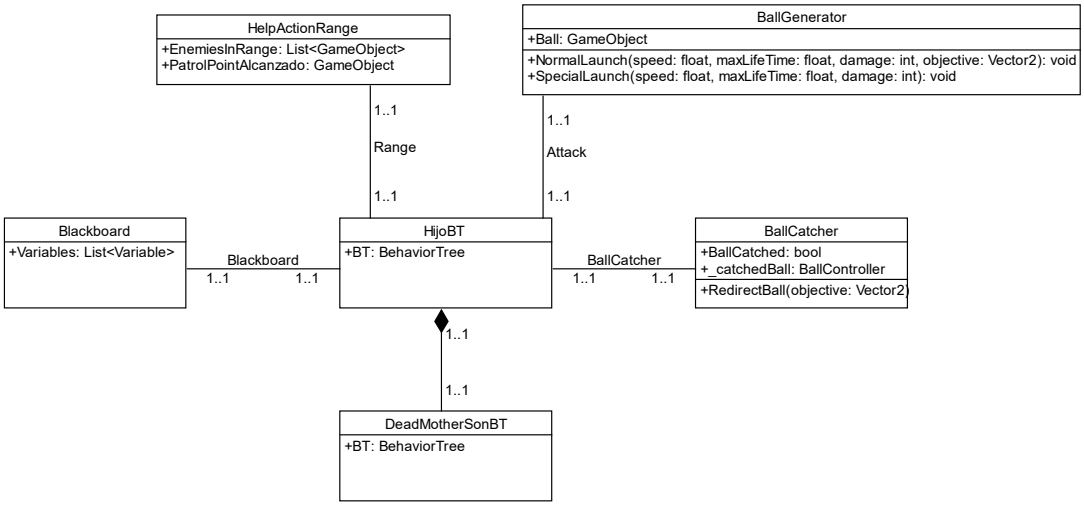


Figura 49: IAComponent del Hijo

En cuanto a la pelota, su IAComponent sigue la siguiente estructura, guardando ciertas similitudes con la estructura del dron especial del Nerd.

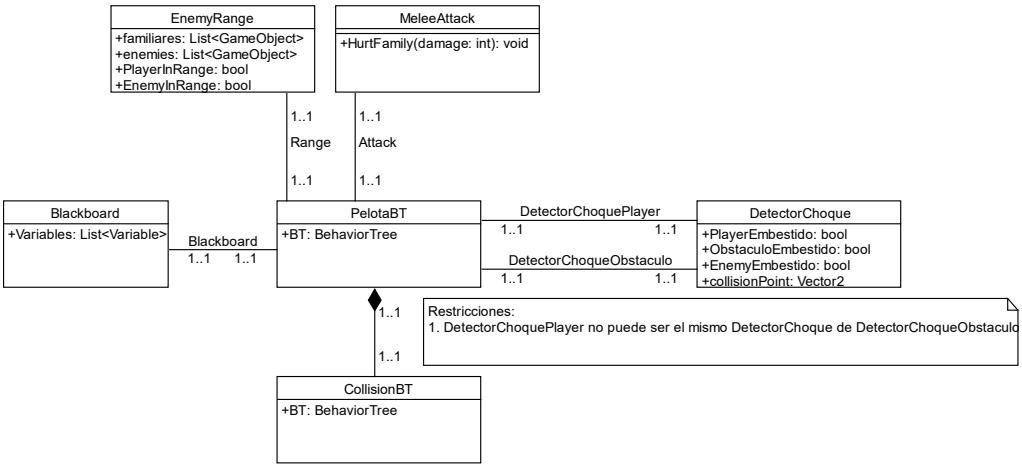


Figura 50: IAComponent de la pelota

4.3. Tecnologías utilizadas

A continuación, trataremos las tecnologías o herramientas de las que se han hecho uso para llevar a cabo el proyecto.

4.3.1. Unity

Unity es, junto a Unreal Engine, uno de los motores de videojuegos más populares en la industria, ampliamente utilizado sobre todo por estudios independientes. Entre los títulos desarrollados con Unity encontramos Monument Valley, Ghost of a Tale o Hollow Knight.

El lenguaje que se utiliza en Unity es C#, lenguaje orientado a objetos y con seguridad de tipos basado en la familia de lenguajes C, basado en la arquitectura .NET [23].

Entre las diversas utilidades que ofrece Unity encontramos su motor de renderizado 3D/2D, su motor de físicas 3D/2D, su motor de animación, su motor de sonido, herramientas de navegación NavMesh, soporte para el multijugador, etc. Entre las principales ventajas que ofrece dicho motor hay que destacar la existencia de una rica documentación, ya sea a través de canales oficiales como su manual de usuario o su API, o a través de los foros, páginas, blogs, etc. realizados y mantenidos por su activa y numerosa comunidad de usuarios.

Unity también cuenta la Unity Asset Store, una extensa tienda que permite adquirir distintos elementos que pueden ser útiles para el desarrollo de los proyectos (como plugins, librerías, modelos, animaciones, etc.) tanto gratuitos como de pago [21][22].

4.3.1.1. Características de Unity

Escenas en Unity

Tanto en Unity como en la mayoría de los motores de videojuegos, una de las piezas fundamentales son las escenas. Dichas escenas representan entornos (los niveles del videojuego, por ejemplo) o menús dentro del juego. Para la creación y edición de escenas, Unity cuenta con una ventana en la que se muestra la escena en la que se trabaja, permitiendo añadir, modificar y/o quitar elementos en la escena.

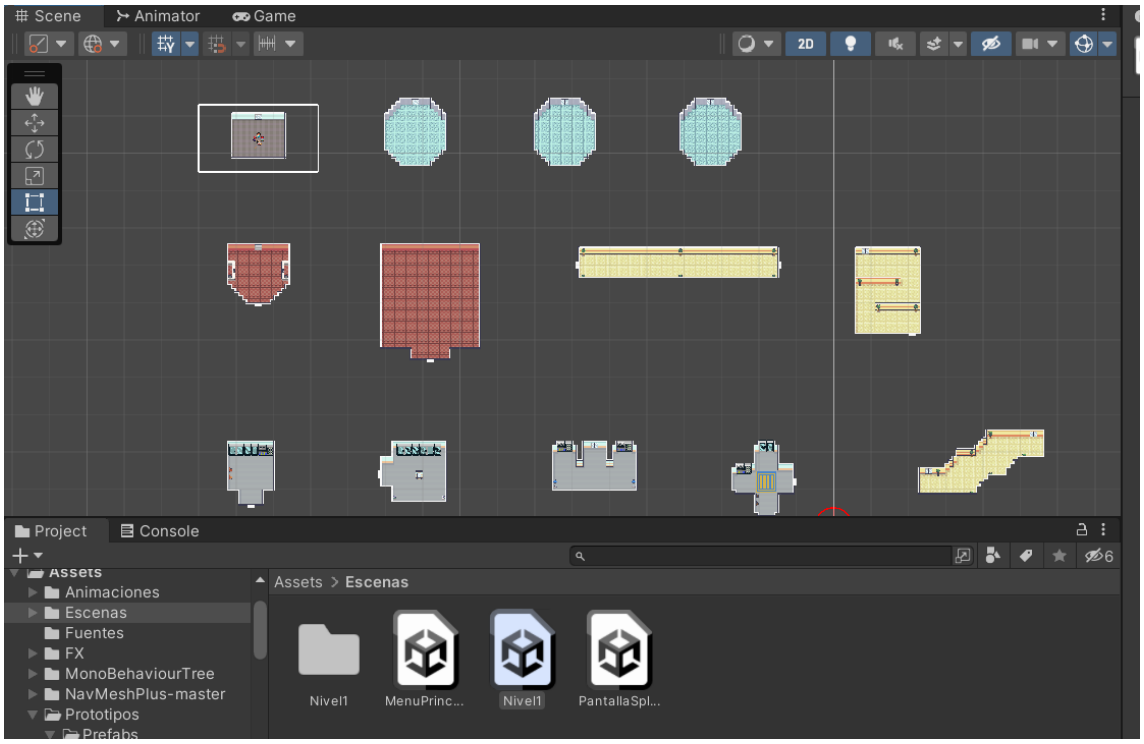


Figura 51: Ejemplo de escena en Unity

GameObjects

Como dice el nombre, un GameObject es un objeto o elemento del juego, pudiendo representar un personaje del juego, un objeto coleccionable, efectos, etc. Los GameObjects funcionan como contenedores de componentes, siendo dichos componentes los que les otorgan sus propiedades.

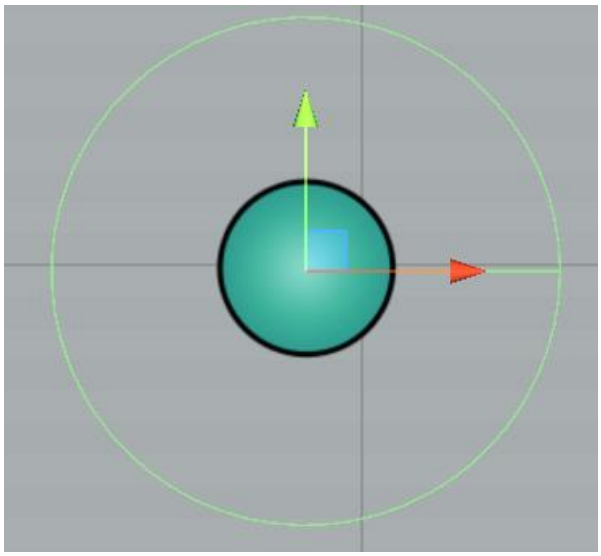


Figura 52: Ejemplo de GameObject en Unity

Jerarquía

Unity cuenta con una ventana en la que se muestran los distintos GameObjects de la escena en la que se está trabajando, organizados de forma jerárquica mostrando las relaciones

entre ellos. La ventana de jerarquía permite acceder a los GameObjects de la escena de la misma forma que si se hiciera click sobre el GameObject en cuestión dentro de la escena, lo cuál puede ser útil cuando la escena crece y se torna difícil encontrar un determinado GameObject.

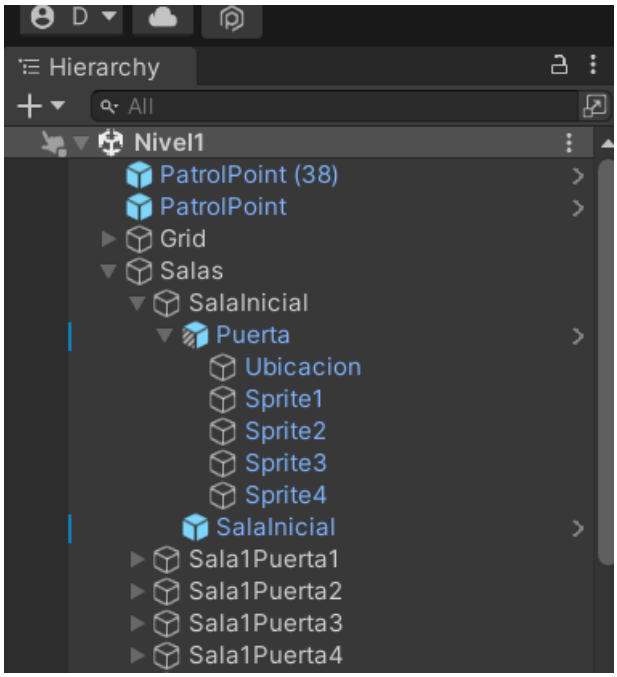


Figura 53: Ventana de jerarquía de Unity

Inspector

La ventana del inspector sirve para revisar los componentes de un GameObject. Desde el inspector es posible visualizar y modificar las características de los componentes, además de poder agregar o eliminar componentes al GameObject.

Figura 54: Inspector de Unity

Scripting

Unity cuenta con una amplia selección de componentes ya implementados que se pueden usar a la hora de desarrollar un videojuego, pero eso no es suficiente. Por ello, Unity permite crear nuevos componentes mediante scripts escritos en C#, que luego se pueden añadir a los GameObjects como cualquier otro componente.

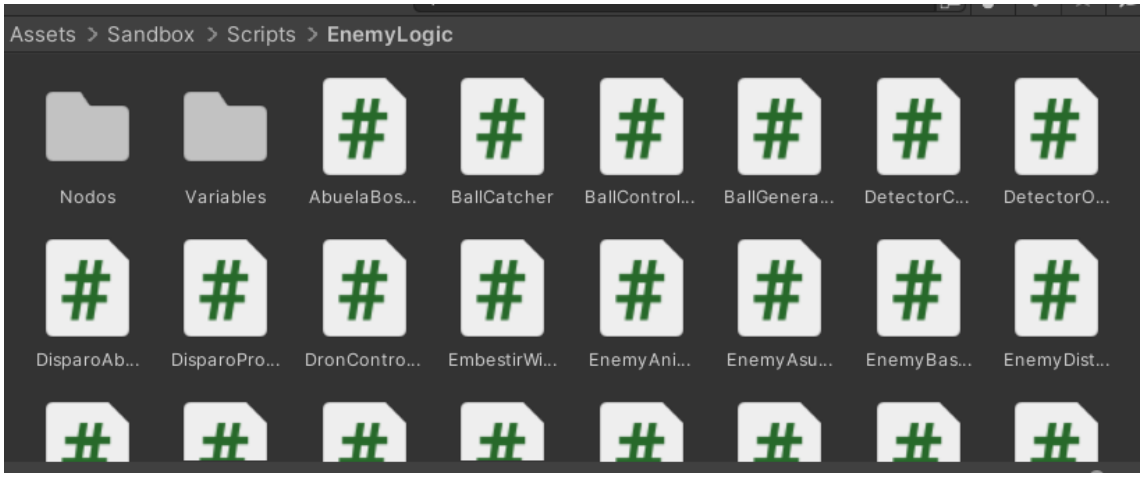


Figura 55: Scripts en Unity

Prefabs

Para poder usar un mismo GameObject en más de una escena, Unity ofrece el sistema de Prefabs, siendo un Prefab un GameObject convertido en un elemento reusable para cualquier escena. Para crear un Prefab, basta con crearlo desde cero en la ventana del proyecto (donde se muestran todas las carpetas y archivos que componen el proyecto) o arrastrando un GameObject desde una escena hasta dicha ventana.

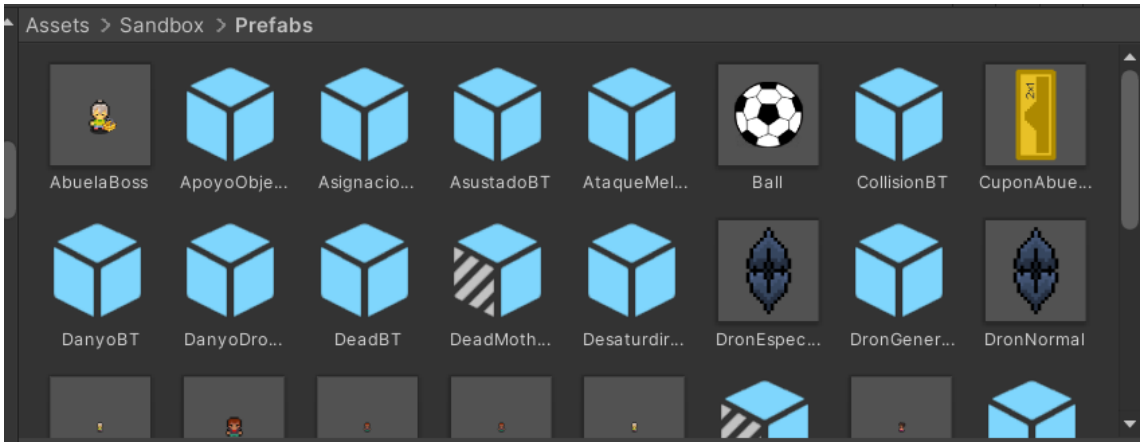


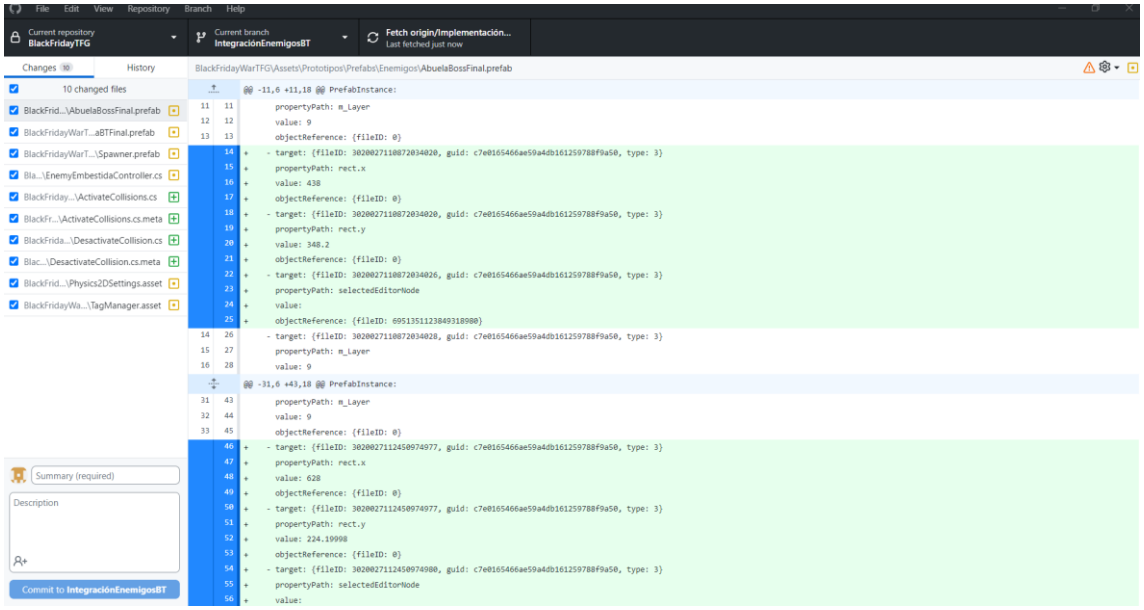
Figura 56: Prefabs en Unity

4.3.2. GitHub

Otra herramienta que debemos mencionar es GitHub, uno de los sistemas de control de versiones basado en Git más utilizados.

GitHub permite alojar código en la nube, registrando los cambios para poder administrar las distintas versiones del proyecto. Esta plataforma es útil sobre todo a la hora de trabajar en equipo en un mismo proyecto. También sirve para evitar pérdidas en el caso de que se pierda el proyecto por un fallo técnico o humano, pudiendo recuperarlo de GitHub.

Cuenta además con GitHub Desktop, una aplicación que permite interactuar con GitHub a través de una interfaz gráfica, lo que facilita el uso de GitHub [24][25].



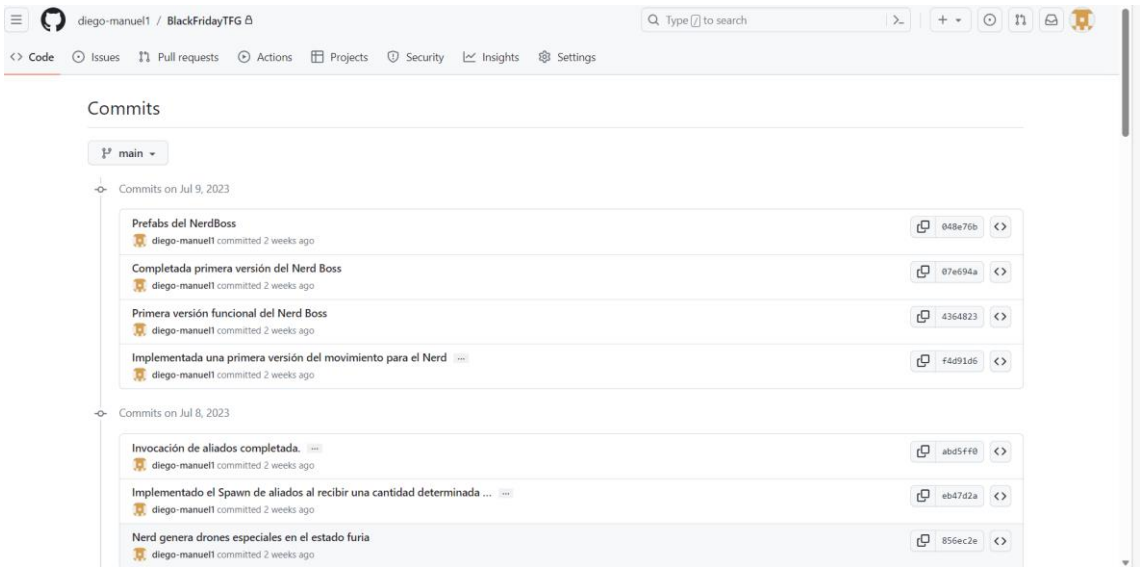


Figura 57: GitHub Desktop y la página de GitHub

4.3.3. Lucidspark

Para realizar los diagramas que representan los Behavior Trees se ha hecho uso de la herramienta web Lucidspark.

Dicha herramienta permite la creación de mapas, esquemas o diagramas mediante el uso de distintas formas y relaciones, pudiendo exportarlo en distintos formatos, incluyendo el formato SVG [26].

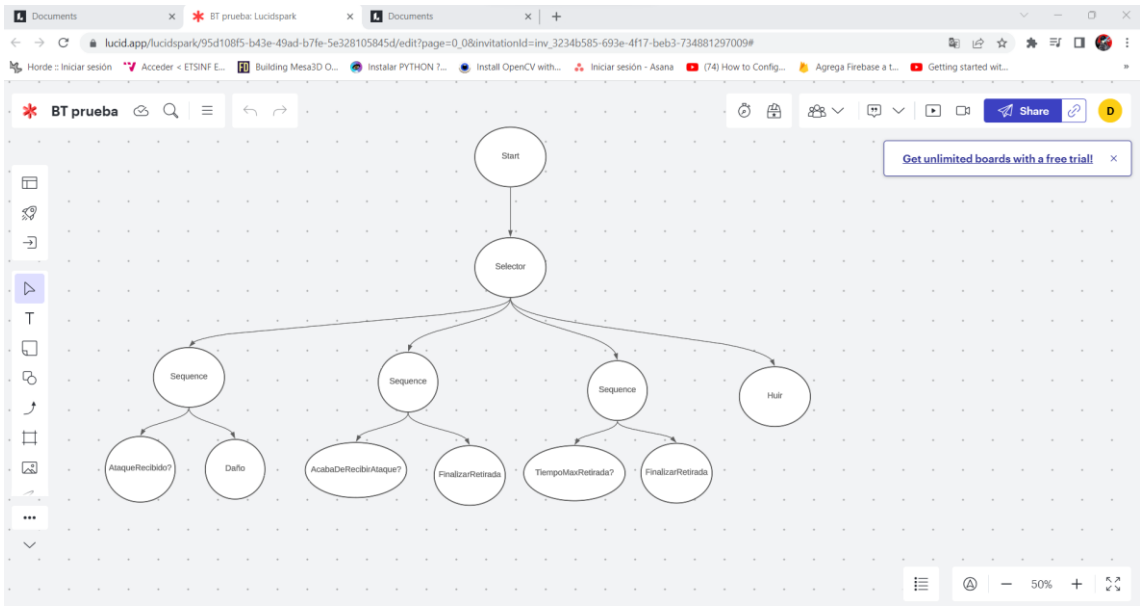


Figura 58: Diagrama en Lucidchart

4.3.4. Jira

Como se mencionó en el plan de trabajo, se haría uso de una aplicación web para la gestión y seguimiento del desarrollo del proyecto. La aplicación elegida para dicha tarea ha sido Jira.

Jira es una herramienta de gestión de proyectos ágiles, ofreciendo, entre otras cosas, cronogramas y tableros Kanban para seguir y organizar las tareas del proyecto. También ofrece

cierta personalización, permitiendo al usuario crear sus propias etiquetas, tipos de tareas, campos, etc. [27].

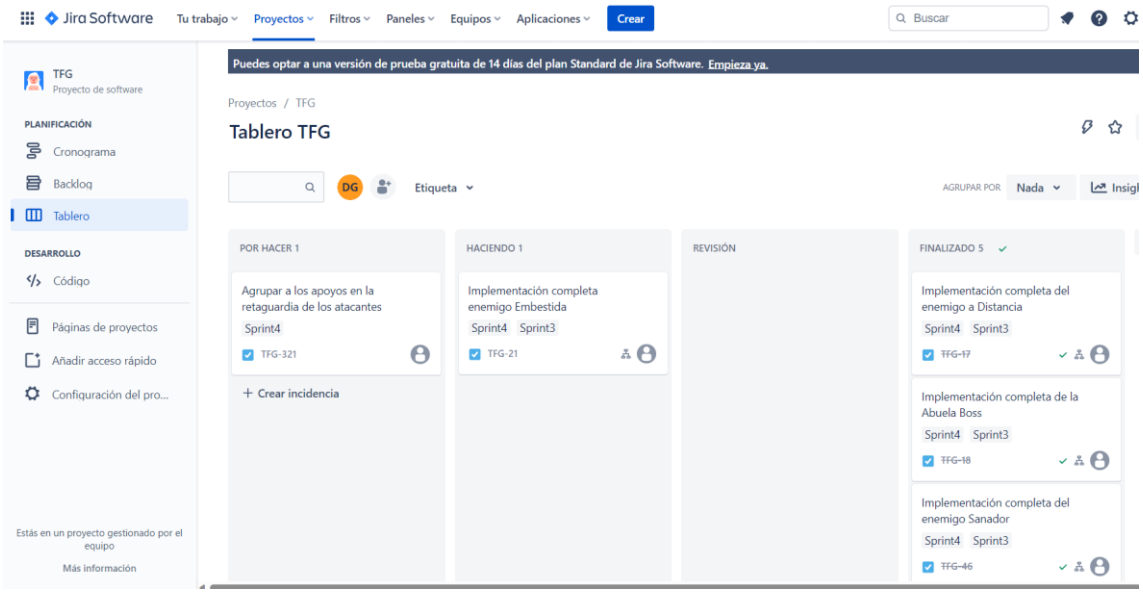


Figura 59: Jira

5. Desarrollo

5.1. Comienzo del proyecto

El proyecto inicial surgió durante la asignatura de Desarrollo de Videojuegos 2D, en el segundo semestre del curso 2022/23, asignatura en la que era necesario desarrollar un videojuego como trabajo final. Inspirándome en el Black Friday (el evento anual originario de los EE. UU. que se considera el inicio de las compras navideñas caracterizado por los grandes descuentos y ofertas de las tiendas) y las parodias de las que ha sido objeto este evento, surgió la temática y las primeras ideas para este videojuego Roguelike. Tras la presentación del pitch-doc, en el que se presentó la primera idea para Black Friday War, se inició el desarrollo del proyecto inicial junto con mis compañeros Francisco José Noguera Guijarro y Jorge Duart Marzo, con quiénes se refinaron y ultimaron los últimos detalles de Black Friday War.

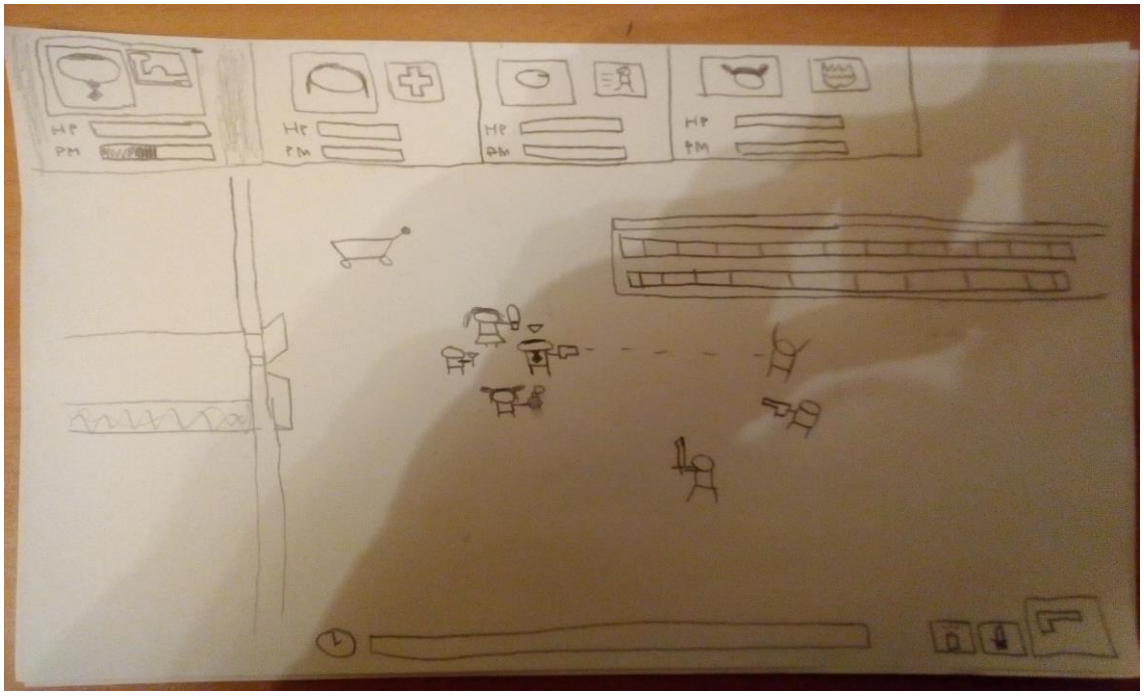


Figura 60: Boceto de Black Friday War

Originalmente estaba realizando un proyecto de emprendimiento para el TFG, pero tras el abandono del líder del equipo, dicho proyecto quedó cancelado. Ante esta situación, finalmente decidí usar el proyecto desarrollado para la asignatura de Desarrollo de Videojuegos 2D para mi TFG, centrándome en la parte de los enemigos, de la cuál era el principal responsable. Tras hablar con el profesor de la asignatura y pedir permiso a mis compañeros de grupo, se registró el nuevo TFG y se inició su desarrollo.

5.2. Desarrollo de los enemigos

En este apartado se recogen los aspectos principales de los enemigos desarrollados, que son los enemigos atacantes, los enemigos apoyo y los jefes. También se recogen detalles de los proyectiles de los jefes del segundo y tercer nivel (los drones del Nerd y las pelotas de la Madre e Hijo), proyectiles que, al contar con un funcionamiento más complejo que el de los proyectiles normales, se decidió modelar su comportamiento usando BTs.

5.2.1. Librerías

5.2.1.1. MonoBehaviourTree

Tras escoger la técnica de los Behavior Trees para la implementación de la inteligencia artificial de los enemigos, se realizó una exhaustiva búsqueda de librerías y plugins para el uso de Behavior Trees. Se encontró un buen número de alternativas, por lo que se decidió estudiar cada una para seleccionar aquella librería que se adaptara mejor a nuestros intereses.

Para ello se creó un proyecto de prueba, en el que se implementaron una serie de Enemigos básicos usando cada una de las librerías encontradas, comparando así los diversos aspectos de su uso, haciendo especial énfasis en la facilidad de uso de la solución, su integración con Unity, si la librería es extensible, si permite la selección aleatoria de comportamiento, las facilidades ofrecidas para la depuración del BT, la documentación de la solución, si ofrece alguna utilidad para la creación de IA táctica y si tiene un editor visual.

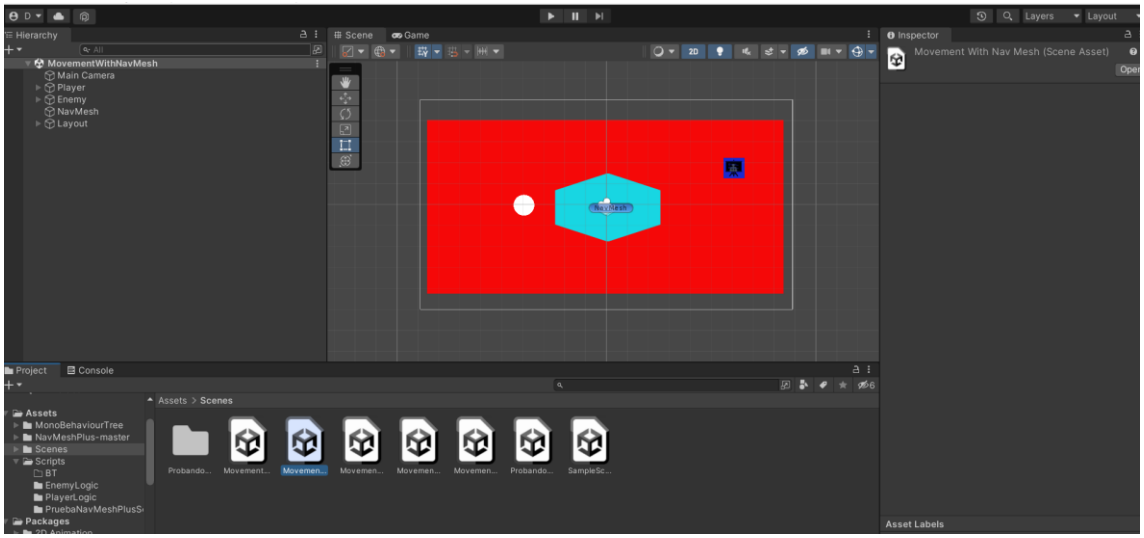


Figura 61: Proyecto de prueba de BT

Los resultados del estudio de las librerías pueden verse en la siguiente tabla:

Tabla 4: Resultado del estudio de librerías para BT

Librería	Facilidad de uso	Integración con Unity	Extensibilidad	Selección aleatoria de comportamiento	Fácil de depurar, testear…	Reusabilidad	Buena documentación	Utilidades para IA táctica	Editor visual
BehaviourBricks	Parcial	Sí	Sí	Sí	Sí	Sí	Correcta, pero escasa	No	Sí
PandaBT	Sí	Sí	Sí	Sí	Sí	Parcial	Sí	No	No
Active Logic: Behavior Trees	Parcial	Sí	¿?	No	Sí	Sí	Sí	No	No
NPBehave	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No
Mono Behaviour Tree	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Fluent-behavior-trees	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No
Xas Fluent Behavior Tree Core	Sí	Sí, aunque el proceso es difícil	Sí	No	No	Parcial	No	No	No

Finalmente, tras el estudio de las librerías, se decidió utilizar la librería MonoBehaviourTree.

5.2.1.2. NavMeshPlus

En el proyecto original, el movimiento de los enemigos se realizó mediante la manipulación del Transform, componente que tienen todos los GameObjects de Unity. Dicho componente contiene la información sobre la posición, rotación y escala del GameObject, pudiendo modificarse a través del inspector de Unity o a través de código.

En la siguiente imagen puede observarse un fragmento del código original de las primeras versiones de los enemigos, en el que se puede observar cómo se consiguió que el enemigo se moviera hacia el jugador variando la propiedad position del Transform del enemigo.

```
1 referencia
private void perseguirJugador(){//Se mueve hacia el jugador
    direccionAnimation();
    vectorJugador = new Vector2(posJugador.position.x, posJugador.position.y);
    transform.position = Vector2.MoveTowards(transform.position, vectorJugador, speed * Time.deltaTime);
}
```

Figura 62: Función perseguirJugador

Con este código, se consigue que el enemigo se mueva en línea recta hacia el punto en el que se encuentra el jugador. El problema de esto es que, si existe un obstáculo en medio del camino, como una pared, el enemigo se quedará atascado tratando de atravesar dicho obstáculo. Para evitar este tipo de situaciones, Unity ofrece solución más sofisticada para el movimiento de entidades: la Navigation Mesh o NavMesh.

El NavMesh podría definirse como una especie de mapa que indica las superficies de la escena de Unity por las que se puede "caminar". Tras generar el NavMesh de la escena, para que una entidad pueda desplazarse a través de la escena solamente necesita el componente NavMeshAgent, el cual calcula la ruta hacia el objetivo de la entidad a través de la superficie "caminable" y desplaza a la entidad. Así, si existe un obstáculo entre la entidad y su objetivo (es decir, una superficie "no caminable"), la entidad puede esquivar dicho obstáculo y llegar sin ningún problema al objetivo.

Por desgracia, las herramientas de NavMesh de Unity parecen estar dirigidas únicamente a proyectos 3D, siendo incapaz de generar una NavMesh para una escena de Unity en 2D. Por ello, se realizó una búsqueda por internet para encontrar alguna solución a esta situación, encontrándose NavMeshPlus.

5.2.2. Componentes

Para conseguir el comportamiento deseado de los enemigos se han tenido que implementar una serie de componentes de diversa índole.

A continuación, detallaremos los más importantes.

5.2.2.1. EnemyRange

Como dice el nombre, este componente representa el rango de acción del Enemigo. Debe colocarse en un GameObject que cuente con un collider de tipo trigger para poder funcionar. En la mayoría de los casos se coloca en un GameObject hijo del Enemigo.

Mediante el uso del collider, EnemyRange detecta cuando un personaje del jugador o un enemigo atraviesa el collider, es decir, detecta cuando se encuentran dentro del rango, registrando los personajes o enemigos que se encuentran en él. Esta información es utilizada de distintas formas.

En el caso de los atacantes y la Abuela, el EnemyRange suele representar el rango de ataque. Por lo tanto, es consultado para saber cuándo es posible realizar el ataque y a que personajes puede atacar (por esa razón se lleva un registro de los personajes del jugador). En el caso de la Abuela se utiliza para saber cuándo puede realizar su ataque a melee y saber a qué personaje del jugador dañar.

La razón de llevar también un registro de los enemigos en rango es que este componente también es usado por los enemigos de apoyo. En dicho caso, el EnemyRange representa el rango de acción del enemigo de apoyo para poder sanar o animar al enemigo asustado. Para ello usan el EnemyRange para saber cuándo tienen al enemigo asustado que buscan en el rango y poder aplicarle la sanación o la animación, dependiendo del tipo de apoyo del que se trate. La Madre e Hijo también lo utilizan de forma similar a los enemigos de apoyo durante la preparación del estado protección, para saber cuándo están lo suficientemente cerca para iniciar el estado protección.

Por otra parte, también es utilizado por los drones del Nerd y las pelotas de la Madre e Hijo para dañar al jugador al alcanzarle.

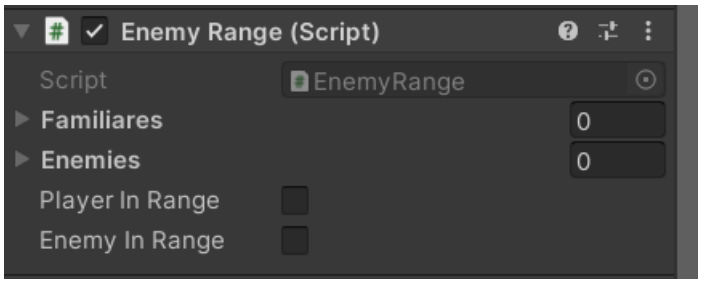


Figura 63: Componente EnemyRange

5.2.2.2. EnemyTerrifiedDetectRange

Este componente es similar en funcionamiento al EnemyRange. Al igual que el EnemyRange necesita encontrarse en un GameObject con un collider trigger para poder funcionar.

En este caso, este componente es exclusivo de los enemigos de apoyo, representando su rango de detección para encontrar enemigos asustados. Cuando un enemigo entra en el collider, se comprueba si se encuentra en el estado asustado, en cuyo caso se registra en una lista que lleva el componente.

Este componente es consultado por los enemigos de apoyo para asignarse un enemigo asustado al que atender.

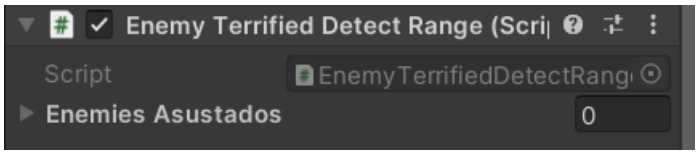


Figura 64: Componente EnemyTerrifiedDetectRange

5.2.2.3. DisparoProyectil

Este componente, como dice el nombre, es el utilizado por el Enemigo para lanzar proyectiles.

Dicho componente, a partir del prefab del Proyectil, genera un proyectil que será lanzado contra la posición en la que se encuentra el jugador en ese momento. Al lanzar el proyectil, se reproduce un sonido representando el sonido de un lanzamiento.

El principal usuario de este componente es el Enemigo distancia, pero también es utilizado por el dron especial del Nerd para lanzar proyectiles.

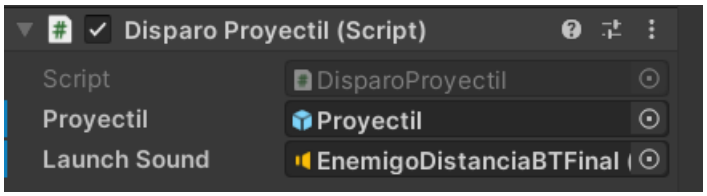


Figura 65: Componente DisparoProyectil

5.2.2.4. EmbestirWithRigidBody

Este componente es exclusivo del Enemigo Embestida. Su función es simplemente conseguir el movimiento de la embestida del Enemigo.

Para ello, este componente mueve en línea recta a una velocidad dada al Enemigo en la dirección en la que se encontraba el jugador al iniciarse la embestida. Para obtener este movimiento se desactiva el NavMeshAgent del enemigo y se manipula directamente al componente Rigidbody del Enemigo.

Figura 66: Componente EmbestirWithRigidBody

5.2.2.5. MeleeAttack

Este componente es el responsable del ataque cuerpo a cuerpo (ataque melee) del Enemigo Melee, aunque también es utilizado por los drones del Nerd y las pelotas de la Madre e Hijo para dañar al jugador al contacto.

Dicho componente, al ser activado para realizar un ataque, consulta el EnemyRange del Enemigo para saber a qué personajes del jugador puede atacar, seleccionándose siempre el primero de la lista del EnemyRange (el que se considera más cercano). Tras ello, se efectúa el ataque aplicando daño al jugador.

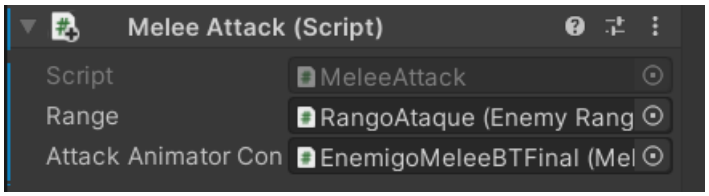


Figura 67: Componente MeleeAttack

5.2.2.6. DetectorObstaculos

Este es un componente común al Enemigo Distancia y Enemigo Embestida.

Dicho componente es utilizado junto al EnemyRange para saber si es posible realizar el ataque, considerándose que el ataque es posible cuándo existe una trayectoria limpia (sin obstáculos) entre el Enemigo y el jugador. Para ello se utiliza el Raycast de Unity, que se basa en el lanzamiento de rayos que detectan GameObjects con colliders.

El componente lanza de forma continua un Raycast desde la posición del Enemigo a la posición del jugador, considerando que hay una trayectoria limpia cuando el rayo no colisiona con nada. Desde el componente podrán seleccionarse las capas a través de las que se lanzará el Raycast, es decir, las capas consideradas como las contenedoras de los obstáculos.

Pueden considerarse un total de dos capas, la de los objetos del entorno (capa Obstacles) y la de los enemigos (capa Enemies). Esto es debido a que en el caso del Enemigo Distancia solo se consideran obstáculos los objetos del entorno mientras que el Enemigo Embestida considera los objetos y los enemigos. En el componente del Enemigo Distancia se añadiría únicamente la capa Obstacles, mientras que el Enemigo Embestida añadiría tanto la capa Obstacles como la Enemies.

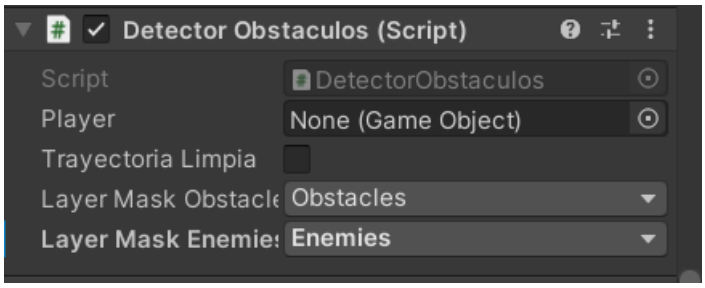


Figura 68: Componente DetectorObstaculos

5.2.2.7. DetectorChoque

Este componente es utilizado principalmente por el Enemigo Embestida. Como dice el nombre se utiliza para detectar choques o colisiones.

Dicho componente se coloca en un GameObject hijo del enemigo, que cuente además con un collider de tipo trigguer. Este componente se utiliza para saber cuándo se ha colisionado contra algún elemento durante la embestida y contra qué a colisionado, distinguiendo entre obstáculo (elemento del escenario), jugador o enemigo. También se utiliza antes de iniciar una embestida para saber si hay obstáculos o enemigos demasiado cerca, en cuyo caso no se realiza la embestida.

También es utilizado por los drones del Nerd y las pelotas de la Madre e Hijo, para saber cuándo colisionan contra algo y saber el qué. En el caso de los drones especiales y las pelotas, también se consulta el punto en el que han colisionado para a partir de él calcular una nueva dirección de movimiento.

Figura 69: Componente DetectorChoque

5.2.2.8. Push

Push es un componente que extiende de MeleeAttack. Su función es realizar el ataque del Enemigo Embestida y el ataque a melee de la Abuela.

Dicho componente distingue dos situaciones, cuando se ataca al jugador y cuando se ataca a un enemigo (recordemos que la embestida del Enemigo Embestida puede dañar enemigos). En el caso del jugador, lo daña y lo lanza contra la pared, mientras que, en el caso de los enemigos, para simplificar, solo reciben daño.

Para empujar al jugador, Push accede a un componente del jugador que fue implementado durante el proyecto original llamado PushingCharacter, que hace que el jugador se mueva en línea recta en una dirección dada hasta colisionar con algo.

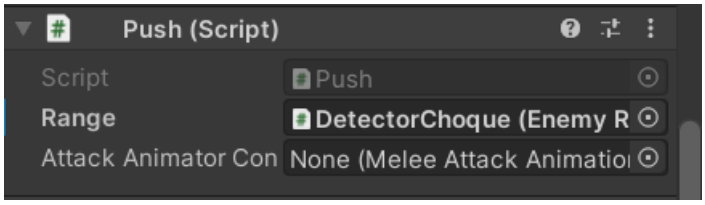


Figura 70: Componente Push

5.2.2.9. DisparoAbuelaSandbox

Como dice el nombre del componente, es el responsable del lanzamiento de proyectiles del jefe del primer nivel (la Abuela).

Dicho componente distingue dos casos: el lanzamiento normal, en el que se lanzan tres proyectiles contra el jugador, y el lanzamiento furia, en el que se lanzan proyectiles en todas direcciones.

Al igual que el componente DisparoProyectil, durante el lanzamiento reproduce un sonido.

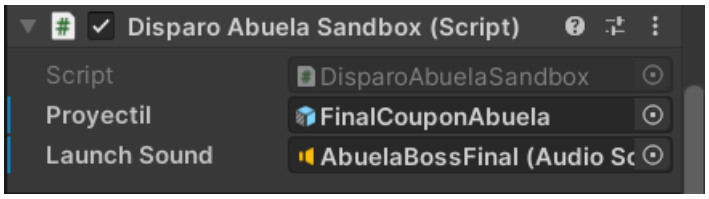


Figura 71: Componente DisparoAbuelaSandbox

5.2.2.10. NormalDronGenerator

Este componente, exclusivo del Nerd, es el responsable de lanzar drones contra el jugador.

El componente es utilizado para generar un dron que es dirigido en la dirección en la que se encontraba el jugador cuando el dron fue generado. Al generarse el dron, se reproduce un sonido de lanzamiento.

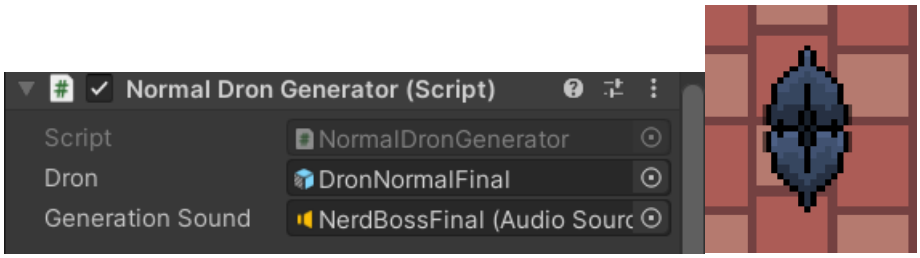


Figura 72: Componente NormalDronGenerator e imagen del dron normal

5.2.2.11. SpecialDronGenerator

Este componente, también del Nerd, es similar al anterior, siendo responsable de la generación de drones especiales.

El componente es utilizado para generar un dron especial, sin ser dirigido contra el jugador. Al generarse el dron, se reproduce un sonido de lanzamiento.

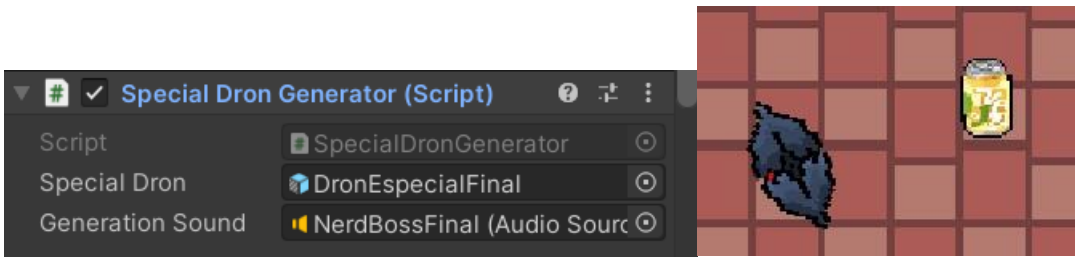


Figura 73: Componente SpecialDronGenerator e imagen de un dron especial

5.2.2.12. OutsideNormalDronGenerator

Extensión del NormalDronGenerator. Representa los generadores externos de drones normales utilizados por el Nerd durante su ataque especial.

El OutsideDronGenerator, al ser activado genera el dron, asignándole un objetivo determinado (normalmente un punto al otro extremo de la sala), pero no activa el dron (se queda inmóvil). Tras su generación, espera que el Nerd le ordene activar el ataque del dron generado, momento en el que el dron se desplazará en línea recta al objetivo asignado.

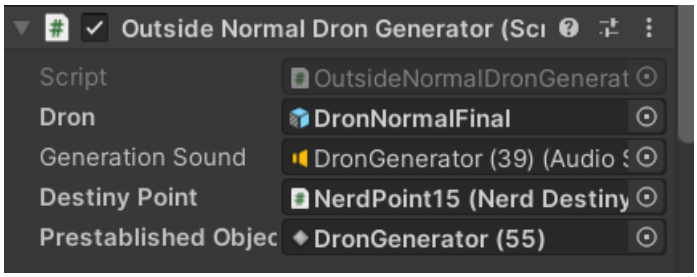


Figura 74: Componente OutsideDronGenerator

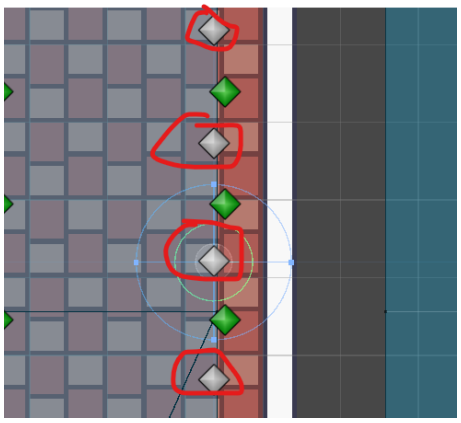


Figura 75: Generadores en la sala del jefe

5.2.2.13. NerdDestinyPoint

Componente que se añade a los puntos clave o destino entre los que se mueve el Nerd. El NerdDestinyPoint hace de intermediario entre el Nerd y los OutsideDronGenerators del extremo en el que se encuentra el punto.

Para ello, el NerdDestinyPoint aplica el patrón Observer. Dicho patrón se basa en definir una dependencia uno a uno entre dos o más objetos para comunicar los cambios sobre un objeto concreto. Para ello, el objeto que sufre cambios (conocido como sujeto) informa al objeto u los objetos que necesitan dicha información (conocidos como los observadores), quiénes previamente se han registran para conocer dicha información [28].

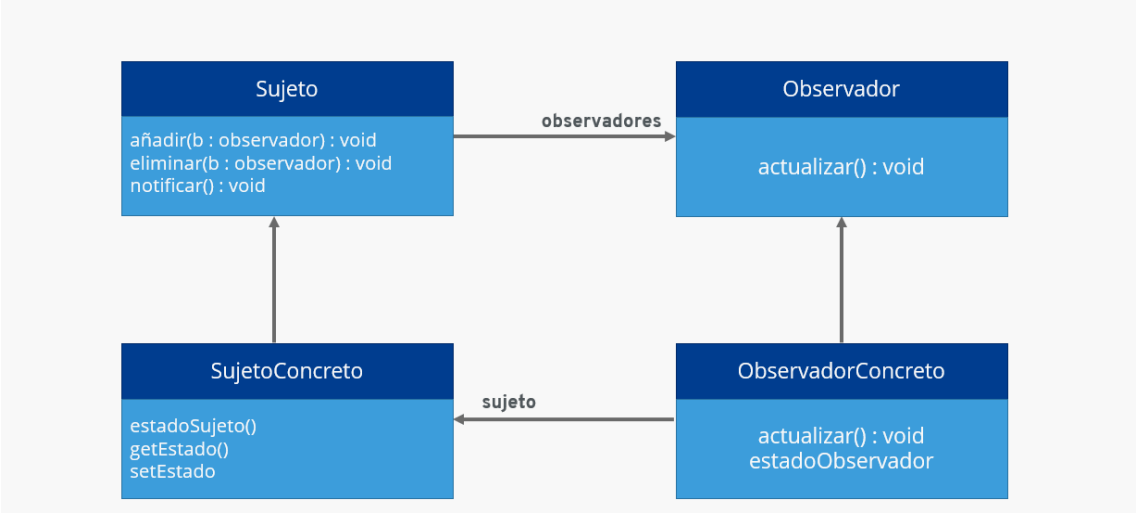


Figura 76: Ejemplo de patrón Observer, IONOS Digital Guide

Unity facilita el uso de este patrón gracias a los Delegates y Events. Un Delegate es una referencia a una función, permitiendo tratar dicha función como una variable y pasarlo para un callback. Al invocarse la función Delegate, se notifica a todas las funciones que hacen referencia al Delegate. El Delegate representaría al sujeto del patrón Observer y las funciones que hacen referencia a dicha función serían los observadores.

Para añadir una capa de abstracción y protección al Delegate, se añaden los eventos [29].

En la siguiente captura de código se observa la función Delegate SpawnNormalDrones y el evento de tipo SpawnNormalDrones, OnSpawnNormalDrones.

```
//Usamos un evento para generar los drones.
public delegate void SpawnNormalDrones(int damage, float speed, bool attackActive);
public event SpawnNormalDrones OnSpawnNormalDrones;
```

Figura 77: Evento OnSpawnNormalDrones

En este caso, el sujeto sería el NerdDestinyPoint y los observadores serían los generadores, quienes se suscriben a los eventos de generación y activación del NerdDestinyPoint para generar y activar los drones.

En el siguiente diagrama se puede observar cómo se realiza la comunicación entre el NerdDestinyPoint y los componentes OutsideNormalDronGenerator, mediante los eventos OnSpawnNormalDrones (para generar los drones) y OnActivateNormalDrones (para activar los drones).

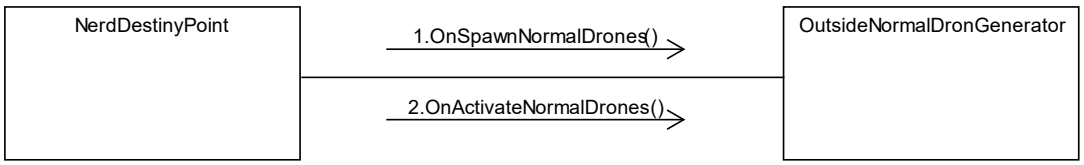


Figura 78: Relación entre NerdDestinyPoint y OutsideNormalDronGenerator

5.2.2.14. NerdPathFinder

Componente responsable del movimiento del Nerd. Dicho componente lleva un registro de todos los puntos clave o destino (DestinyPoints) y puntos intermedios de la sala.

El NerdPathFinder usa el registro para obtener los puntos adyacentes al punto en el que se encuentra el Nerd, considerándose puntos adyacentes aquellos que se encuentran arriba, abajo, a la derecha y a la izquierda (los puntos en diagonal no se cuentan). Para mover al Nerd, el NerdPathFinder selecciona uno de los puntos adyacentes según el tipo de movimiento que realiza.

El NerdPathFinder distingue dos tipos de movimiento: el movimiento aleatorio y el movimiento hacia un punto destino concreto. Durante el movimiento aleatorio el Nerd selecciona aleatoriamente los puntos entre los que se mueve, sin dirigirse a un punto destino concreto.

En el movimiento hacia el punto clave asignado, el Nerd, dado un punto destino concreto, se mueve hasta alcanzar dicho punto, seleccionando para ello los puntos más cercanos a dicho punto destino.

La razón de la existencia de estos dos tipos de movimiento es la siguiente. Si solo se realizara el movimiento aleatorio, el Nerd podría tardar mucho tiempo en alcanzar un punto destino para realizar el ataque especial, o incluso no llegar nunca a un punto destino. En el caso del movimiento hacia un punto destino concreto, como siempre escoge los puntos más cercanos al punto destino, se movería todo el rato por los mismos caminos, resultando muy repetitivo.

Teniendo ambos tipos de movimiento, al combinarlos haciendo que primero se mueva entre puntos aleatorios y, pasado un tiempo, se dirija a un punto destino concreto, conseguimos que el movimiento del Nerd sea más atractivo, sin llegar a quedarse atascado ni volverse repetitivo.

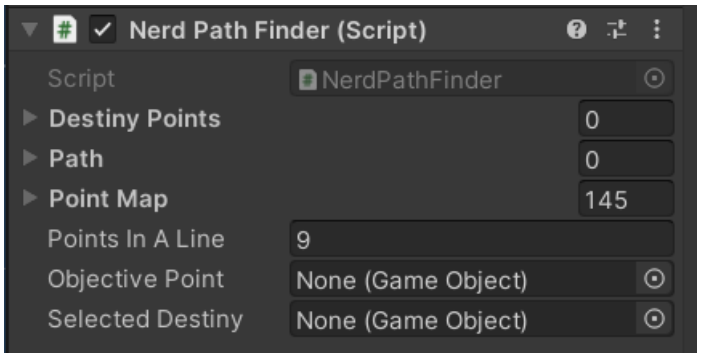


Figura 79: Componente NerdPathFinder

En la siguiente imagen se puede visualizar la sala del jefe con los puntos entre los que se mueve el Nerd, siendo los de color verde puntos intermedios mientras que los rojos (que se encuentran en los extremos) representan los puntos destino.

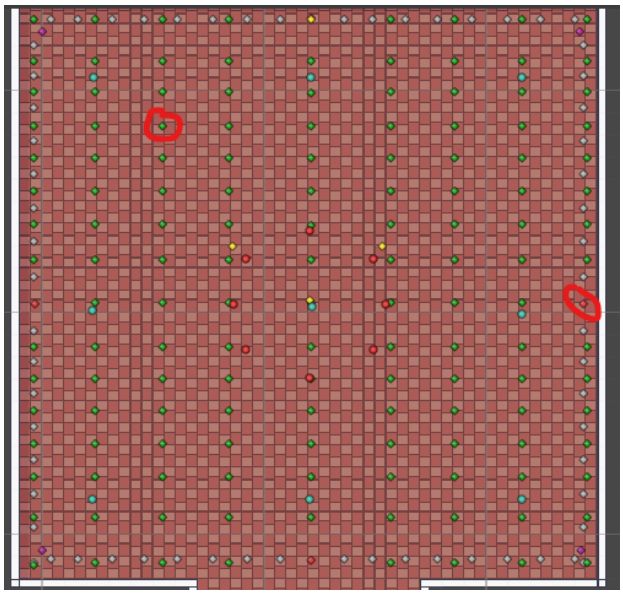


Figura 80: Puntos de movimiento del Nerd

5.2.2.15. NerdPointsDetector

Componente que se coloca en un GameObject hijo del Nerd con un collider de tipo trigger. Este componente sirve para detectar cuando el Nerd alcanza el punto intermedio o punto destino asignado como objetivo.

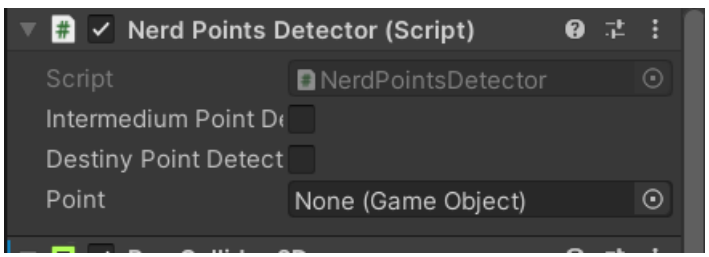


Figura 81: Componente NerdPointsDetector

5.2.2.16. SpawnerNerdAlly

El Nerd tiene la capacidad de generar aliados (enemigos normales) cuando se le reduce una cantidad de vida determinada. El SpawnerNerdAlly es una versión extendida del Spawner (componente responsable de la generación enemigos).

Este componente, que se encuentra en la sala del jefe, tiene la función de generar los aliados del Nerd. Para comunicar cuándo debe generar enemigos, se ha aplicado el patrón Observer usando Delegates y events. El controlador del Nerd (sujeto) produce un evento con el que comunica a los componentes SpawnerNerdAlly (observadores) que deben generar un enemigo.

A su vez, el SpawnerNerdAlly suscribe al controlador del Nerd (lo registra como observador) al evento de muerte del aliado (sujeto). Esto es necesario ya que mientras haya al menos un aliado vivo el Nerd es invulnerable. De esta forma el Nerd es notificado cuando un aliado muere y, cuando no queden aliados, podrá abandonar su invulnerabilidad.

En el siguiente diagrama se puede observar cómo se comunica el controlador del Nerd (NerdBossController) con los generadores mediante el evento OnSpawnAllies. De la misma forma se observa cómo se comunica el controlador del aliado (un componente controlador normal que hereda de EnemyBaseController) con el evento OnDieAlly para notificar la muerte del aliado.

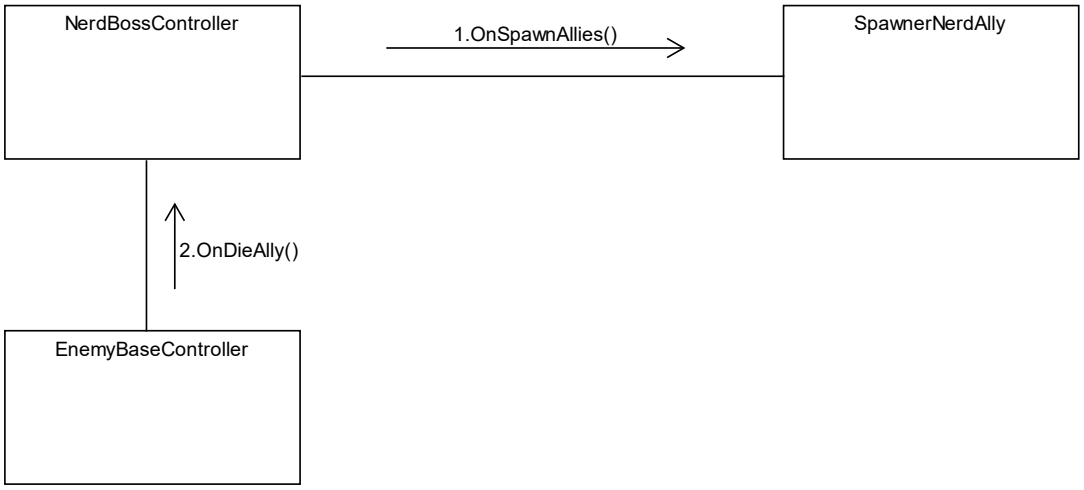


Figura 82: Relación Nerd, generadores y aliados

En las siguientes imágenes se pueden observar los generadores de aliados colocados en la sala del jefe (puntos morados) y la vista de su componente.

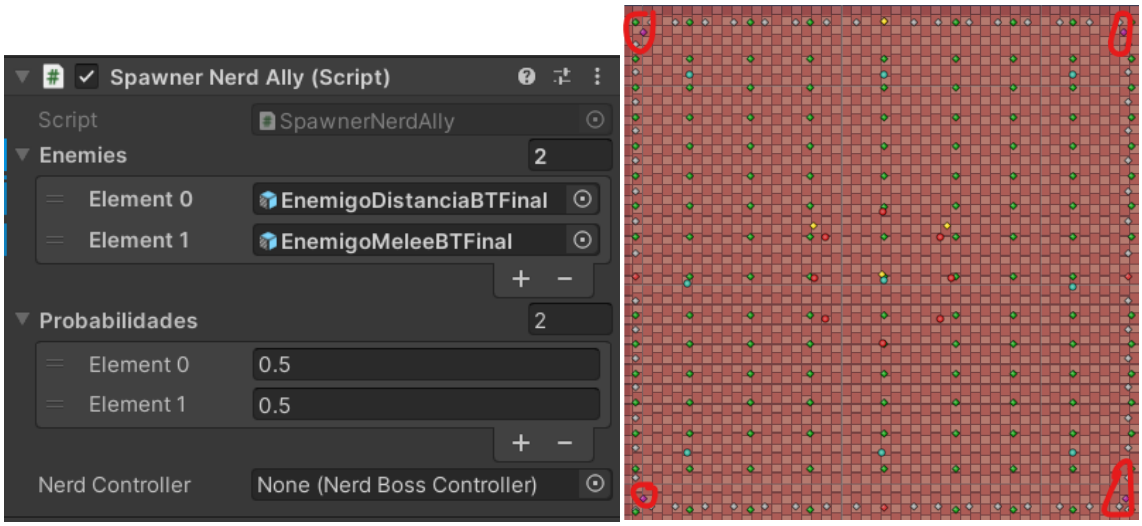


Figura 83: Componente SpawnerNerdAlly y su colocación en la sala del jefe

5.2.2.17. BallGenerator

Este componente es común en la Madre e Hijo. Es el responsable del lanzamiento de los proyectiles sus proyectiles: las pelotas.

Dicho componente distingue dos lanzamientos: el normal y el especial. En el normal, se generan tres pelotas y se lanzan contra el jugador. En el especial, se lanzan proyectiles en todas direcciones, siendo este lanzamiento el utilizado para realizar el lanzamiento especial de la Madre.

Al generarse la pelota o el conjunto de pelotas se reproduce un sonido de lanzamiento.

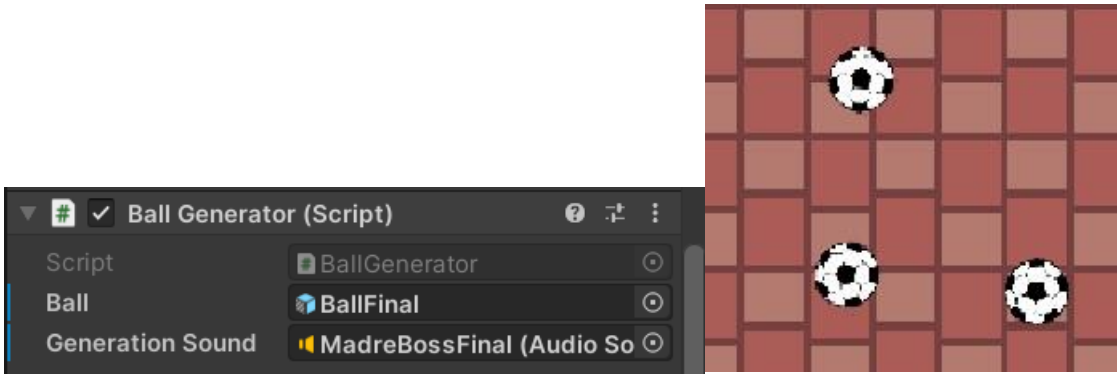


Figura 84: Componente BallGenerator e imagen de los proyectiles generados

5.2.2.18. BallCatcher

Este componente es exclusivo del Hijo. Dicho componente debe colocarse en un GameObject hijo con un collider de tipo Trigger.

El BallCatcher sirve para detectar cuando una de las pelotas lanzadas por la Madre o el Hijo alcanza al Hijo, permitiendo acceder a la pelota para redirigirla contra el jugador.

Este componente es utilizado durante el lanzamiento especial de la Madre y durante el estado furia del Hijo.

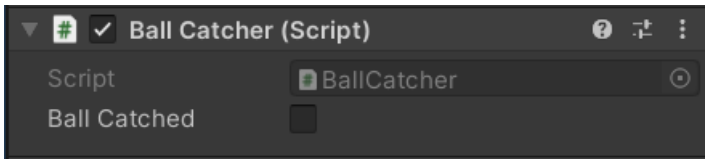


Figura 85: Componente BallCatcher

5.2.2.19. HelpActionRange

Este componente es exclusivo de la Madre e Hijo.

Dicho componente sirve para detectar cuando el Enemigo llega a un punto (PatrolPoint). Este componente fue implementado originalmente para los enemigos de apoyo cuando se planteó la idea de hacer que se movieran entre puntos del escenario realizando patrullas.

Finalmente, el sistema de movimiento a través de puntos de patrulla se descartó para los enemigos de apoyo, pero se decidió reutilizar para el movimiento de la Madre, haciendo que esta se mueva entre puntos del escenario (simulando patrullas), y para el movimiento del Hijo cuando gira alrededor del centro durante los lanzamientos especiales de la Madre.

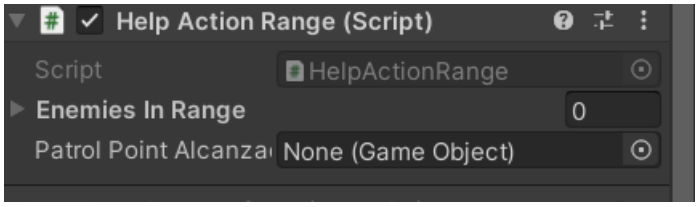


Figura 86: Componente HelpActionRange

En la siguiente imagen se pueden observar los PatrolPoints de la Madre e Hijo en la sala del jefe, siendo los de la Madre aquellos con un color verde azulado y los del Hijo los de color rojo (que se colocan alrededor del centro para que se mueva alrededor de la Madre durante el lanzamiento especial).

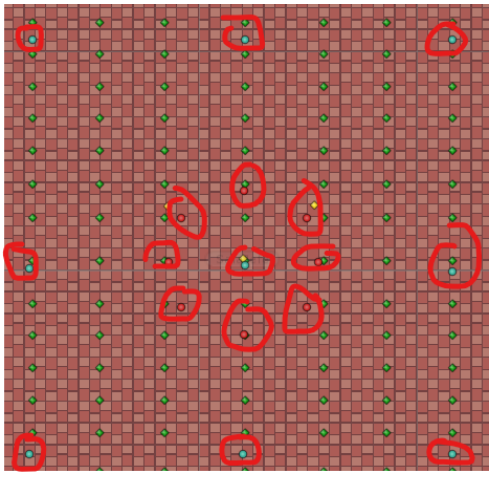


Figura 87: PatrolPoints de la Madre e Hijo

5.2.3. Implementación de los Behaviour Trees

Hasta ahora hemos visto los componentes implementados para realizar las distintas acciones de los enemigos. Ahora bien, por si solos, esos componentes son inútiles y necesitan ser administrados para lograr que los enemigos se comporten tal y como hemos planeado.

Dicha administración la realizan los Behaviour Trees, quienes acceden a dichos componentes, bien para realizar consultas o para manipularlos. Podría decirse que dichos componentes son el cuerpo del enemigo, mientras que el BT sería su cerebro, que coordina las distintas partes del cuerpo para que funcione correctamente.

Para la implementación de los BT se ha hecho uso de la librería mencionada anteriormente: MonoBehaviourTree.

Dicha librería cuenta con un nodo Subtree, que permite conectar un BT a otro BT como un subárbol. Aprovechando esto, se decidió construir todos los BT de los enemigos como composiciones de subárboles.

Figura 88: Ejemplo de BT compuesto por subtrees. Esta BT corresponde al Enemigo Melee

Principalmente fueron dos razones las que nos llevaron a construir los BT de esta forma. La primera fue la facilidad que nos ofrece esta estrategia a la hora de organizar las BT internamente, encapsulando en un subárbol cada estado o rutina del enemigo.

La segunda razón es la de potenciar la reutilización. Muchos enemigos cuentan con algunos comportamientos y rutinas similares, por lo tanto, al organizarse los estados y rutinas en subárboles, dichos subárboles pueden ser utilizados para la construcción de otro BT. En el ejemplo visto en la figura anterior podemos ver el BT del estado asustado del Enemigo Melee: AsustadoBT. Como el estado asustado es igual para todos los enemigos de ataque, dicha BT puede reutilizarse en la construcción de las BTs del resto de los enemigos.

Para poder reutilizar BTs se ha decido aprovechar el sistema de prefabs de Unity, creando prefabs de los GameObjects que contienen la BT reutilizable, permitiendo así que en cualquier momento pueda agregarse dicha BT a una BT en construcción.

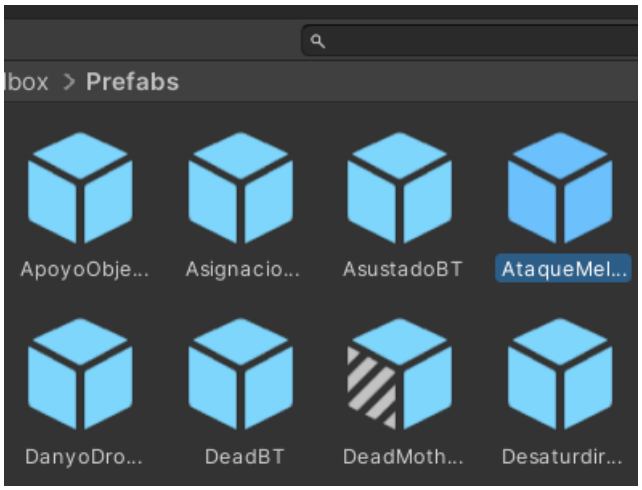


Figura 89: Ejemplo de BTs convertidos en prefabs

Para almacenar las variables necesarias para el correcto funcionamiento de los BT, a todos los enemigos se les agregó un componente Blackboard, para que la BT del enemigo la consulte y modifique según sea necesario.

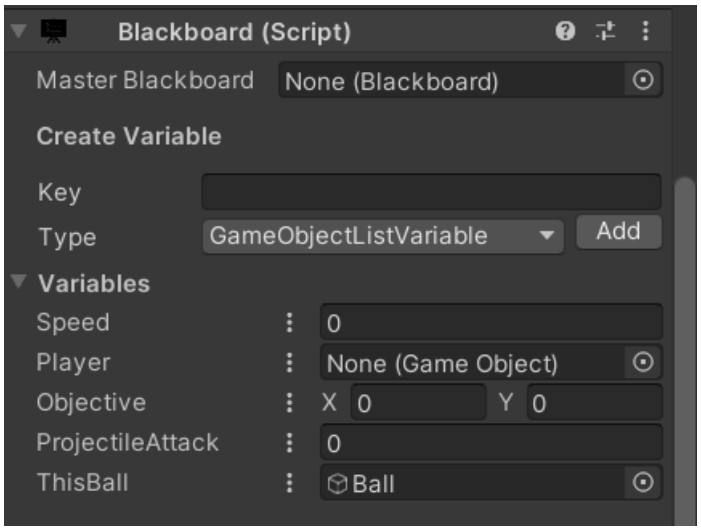


Figura 90: Ejemplo de Blackboard

5.2.4. Controladores de enemigos

Los controladores de los enemigos son unos componentes especiales cuya principal función son inicializar la Blackboard del enemigo en cuestión. También realiza otras funciones, como la de realizar de intermediario con el entorno en ciertas interacciones (al recibir daño, por ejemplo).

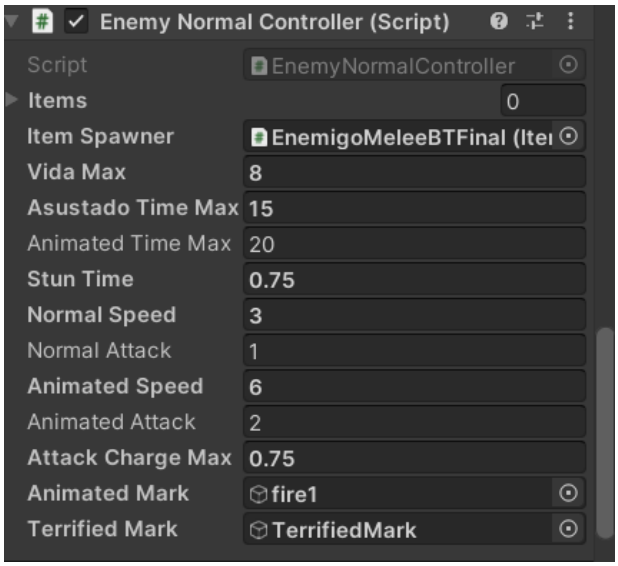


Figura 91: Ejemplo de controlador

También es el responsable de gestionar algunos tiempos de carga de algunas acciones del Enemigo, como puede ser la realización de un ataque. Para dicha gestión, se hace uso de las corrutinas de Unity, las cuáles ejecutan una rutina para contar el tiempo y, al pasar el tiempo necesario, realizan la acción pertinente (un ataque, una sanación, etc).

Para potenciar la reutilización de código, se implementó el EnemyBaseController como clase abstracta de la que extienden los demás controladores, siguiendo el esquema visto en el apartado de diseño. Para los enemigos atacantes tenemos el EnemyNormalController que es la base de los controladores de los enemigos atacantes además de ser el controlador del Enemigo Melee. El controlador del Enemigo Distancia (EnemyDistanciaController) extiende del EnemyNormalController y, en cuánto al del Embestida, debido a algunas similitudes con el Enemigo Distancia, su controlador (EnemyEmbestidaController) extiende de él.

Por otra parte, los enemigos de apoyo, al tener un comportamiento muy similar, utilizan el mismo controlador (HelpEnemyController) que extiende del EnemyBaseController.

Los controladores de los jefes también extienden del EnemyBaseController. En el caso de la Madre e Hijo, el controlador del Hijo (SonBossController), debido a ciertas similitudes con la Madre, extiende del controlador de la Madre (MotherBossController).

En cuanto a los proyectiles del jefe del segundo y del tercer nivel, sus controladores también extienden del EnemyBaseController. En cuánto a los drones, al tener ciertas similitudes el dron especial y el dron normal, el controlador del dron especial (SpecialDronController) extiende del controlador del dron normal (DronController).

5.2.5. Animaciones y efectos

Respecto a las animaciones, para realizar las animaciones de movimiento de los enemigos nuevos se siguió el esquema utilizado para las animaciones de los enemigos del proyecto original, contando con un parámetro AnguloJugador en el controlador de las animaciones para variar la activación de una animación u otra.

Para gestionar las animaciones, se creó un componente EnemyAnimationsManager para los enemigos atacantes y jefes, que a partir de la velocidad del Enemigo y de la posición del jugador,

se varia AnguloJugador para activar la animación de movimiento más conveniente. Por ejemplo, si se mueve hacia la derecha, se activará la animación de movimiento a la derecha. Dicho componente también cambia de posición el arma u objeto que porte el Enemigo según su movimiento.

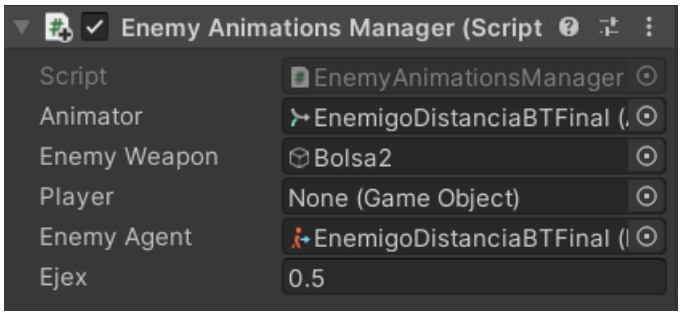


Figura 92: Componente EnemyAnimationsManager

También se implementó un componente HelpEnemyAnimationsManager que extiende del anterior, que actualiza las animaciones según la velocidad del Enemigo únicamente. Dicho componente se dirige a los enemigos de apoyo.

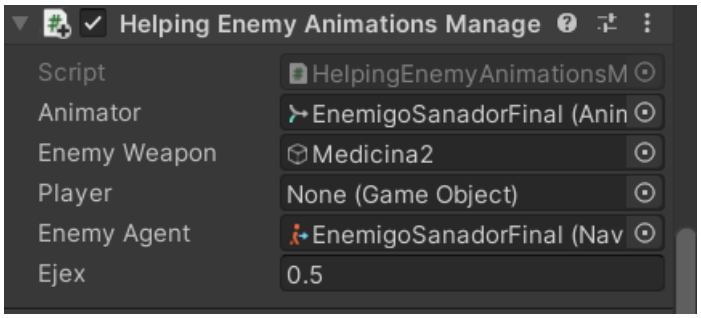


Figura 93: Componente HelpingEnemyAnimationsManager

En cuanto a las animaciones de los proyectiles, para los proyectiles normales (Enemigo Distancia, Abuela y drones especiales) simplemente se agregó un Sprite con un script RotateLata que gira el Sprite. Dicho script también se utiliza para el girar el Sprite de la pelota.

También se debe mencionar la animación de ataque del enemigo a melee, que se hizo utilizando cuatro Sprites animados de un rodillo colocados alrededor del enemigo. Se decidió usar un script que administre, según la posición del jugador, el rodillo animado que se activará. Dicho script es accedido por el componente MeleeAttack.

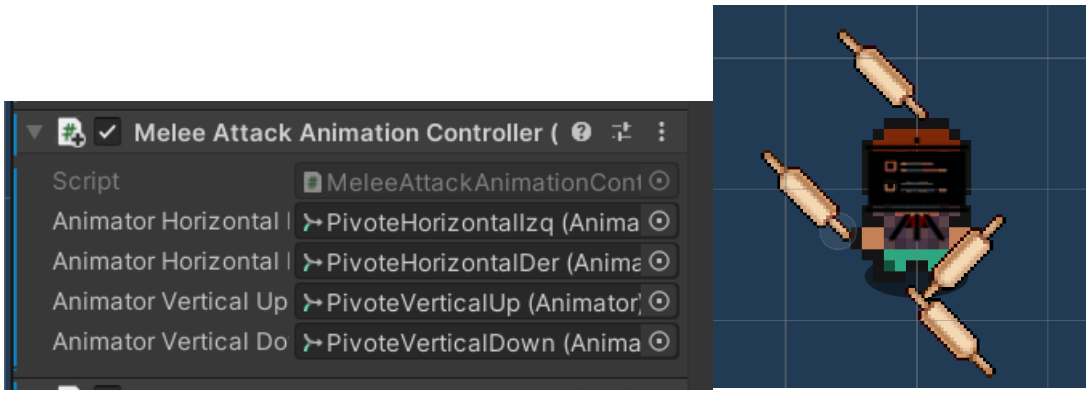


Figura 94: Componente MeleeAttackAnimationController y enemigo con los cuatro rodillos

En cuanto al dron especial, se le agregó un Sprite animado de una especie de dron futurista.

Cuando los proyectiles desaparecen al chocar algo se agregó un efecto de humo que creó mi compañero Francisco durante el proyecto original.

También se tuvo que agregar algunos efectos más a los enemigos para indicar su estado. Para el estado asustado, se añadieron unas gotas sobre la cabeza de los enemigos para simular que sudan mientras que para el estado animado se agregó una animación de fuego para destacar su motivación.

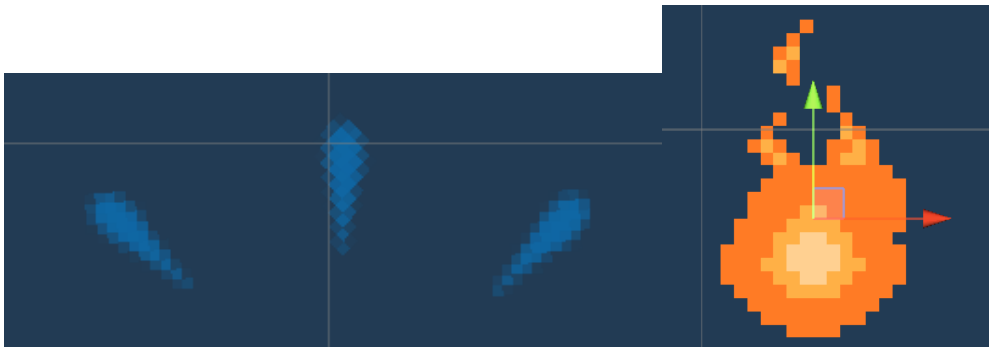


Figura 95: Sprites de las gotas y el fuego

Para el estado aturdido del Enemigo embestida, se decidió colocar sobre la cabeza del Enemigo un sistema de partículas que emite estrellas.

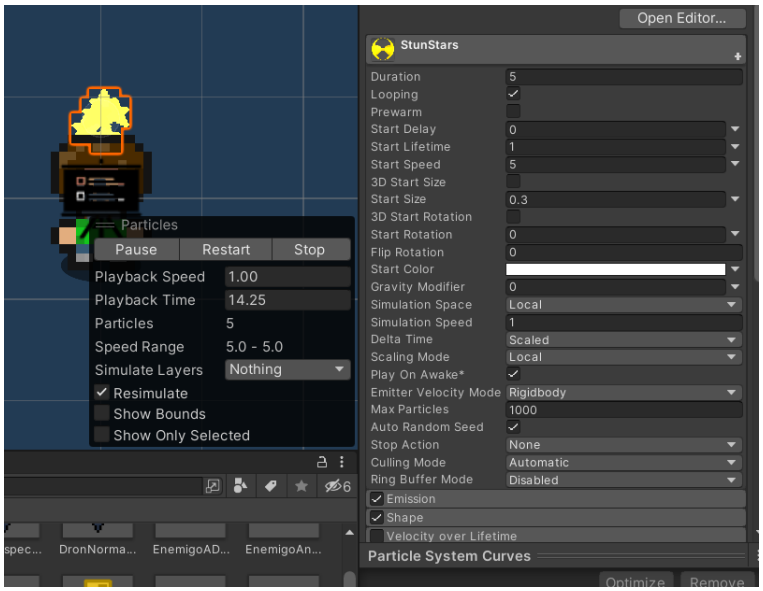


Figura 96: Sistema de partículas del estado aturdido

Para indicar cuando se cura o se anima alguien se decidió cambiar el color del enemigo que está siendo curado o animado a verde o naranja, respectivamente.

5.3. Nivel y control de enemigos

El sistema de administración de nivel y de las salas fue implementado originalmente por mi compañero Francisco para el proyecto original de Desarrollo de Videojuegos 2D. Este punto recoge las modificaciones y cambios que se han realizado sobre dicho sistema para adaptarlo a las nuevas necesidades del proyecto.

5.3.1. Generación de los enemigos

Como ya se mencionó en puntos anteriores, la generación de los enemigos se realiza mediante un componente llamado Spawner, creado para el proyecto original por mi compañero Francisco. Dichocomponente cuenta con una lista de enemigos y una lista de probabilidades, estando cada probabilidad ligada a un enemigo de la lista dependiendo de la posición en la que se encuentren (es decir, la primera probabilidad pertenece al primer enemigo, la segunda al segundo enemigo y así consecutivamente), de tal forma que, al generar un enemigo, si hay, por ejemplo, un enemigo sanador con una probabilidad de 0,5, habrá un 50% de probabilidades de que el enemigo generado sea un sanador.

Dichos Spawners son administrados por los controladores de las salas que componen el nivel (HallController), quiénes tienen una lista con todos los Spawners que se encuentran en la sala.

Figura 97: Spawners de una sala normal

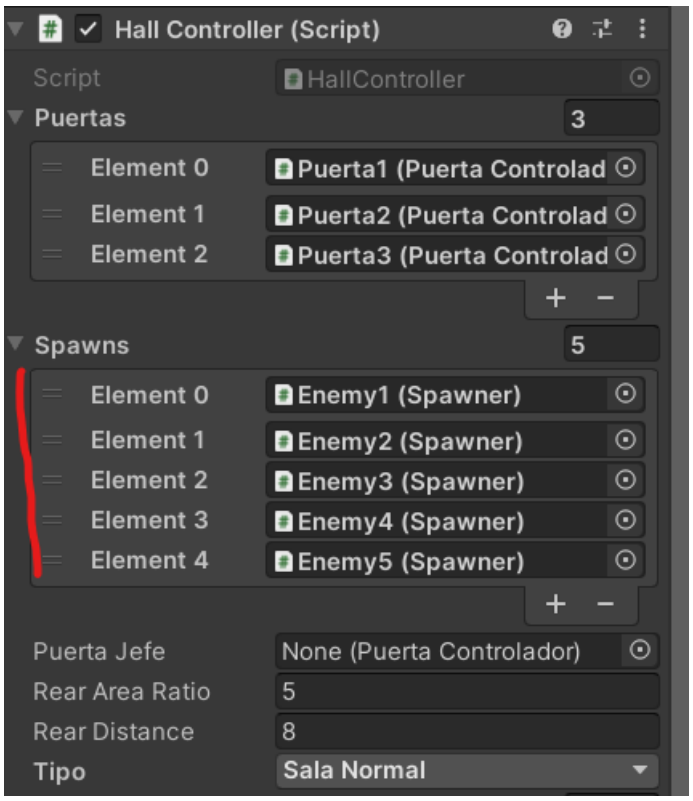


Figura 98: Lista de Spawners de un controlador de sala

Cuando el jugador entra en una sala marcada como SalaNormal, el controladorcierra las puertas de la sala y utiliza los Spawners para generar una oleada de enemigos. Un caso especial es el de la sala del jefe (marcada como SalaJefe), que contiene un Spawner por cada jefe de nivel, habiendo un Spawner para la Abuela, otro para el Nerd, otro para la Madre y otro para el Hijo. La

sala también cuenta con unos Spawner que son utilizados por el Nerd para generar aliados, pero dichos Spawners no son administados por el controlador de la sala.

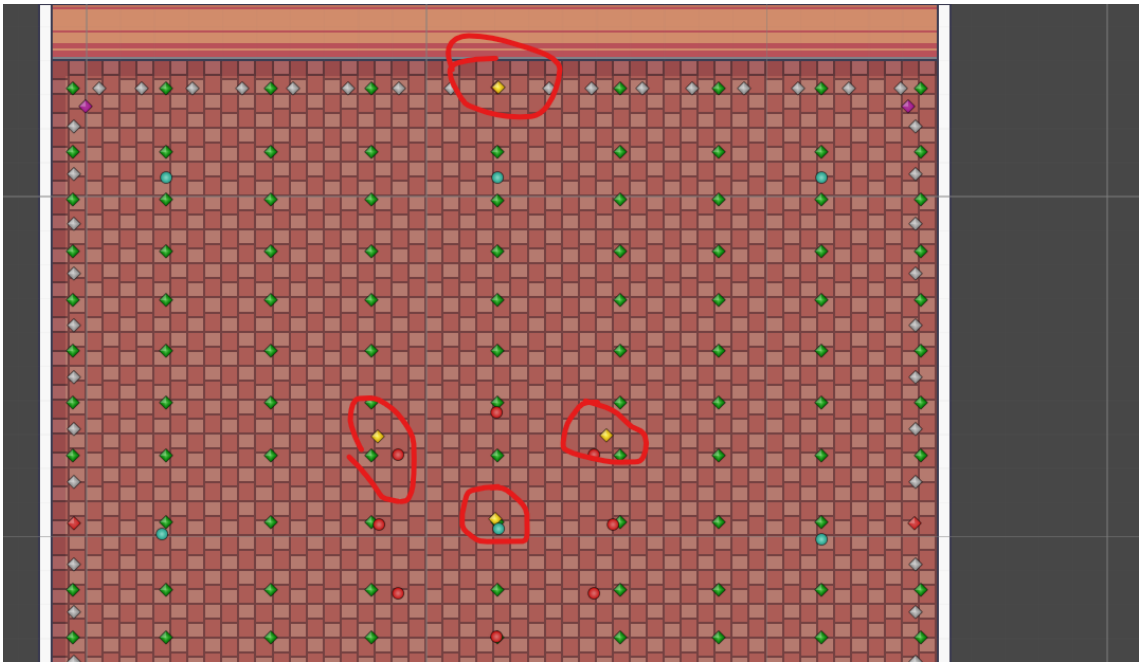


Figura 99: Spawners de la sala del jefe

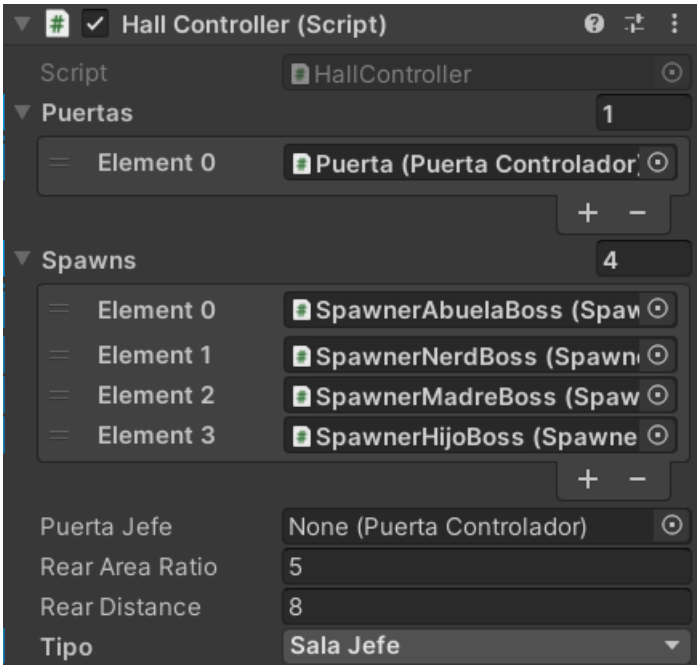


Figura 100: Controlador de la sala del jefe con la lista de Spawners

Durante el proyecto original de Desarrollo de Videojuegos 2D, al contar únicamente con un único jefe (la Abuela) no había ningún problema a la hora de generar al jefe, ya que se trataba solamente de generar una oleada con un único enemigo, pero ahora, al contar con varios jefes, debemos asegurarnos de que se active únicamente el Spawner del jefe correcto y que no se generen más jefes, por lo que la solución original ya no nos sirve. Por esta razón se tuvieron que realizar una serie de modificaciones sobre HallController para que el controlador de sala, al generar los enemigos, distinga los entre el caso en el que debe generar una oleada normal,

activando todos los Spawners, y el caso en el que debe generar un jefe de nivel, activando solo los Spawners relacionados dicho jefe.

El controlador de una sala normal (marcado como SalaNormal) seguirá generando los enemigos como siempre, mientras que el controlador de la sala del jefe (marcado como SalaJefe), cuando llegue el jugador a la sala consultará el nivel en el que se encuentra para saber que jefe toca y activar su Spawner correcto y generar el jefe de dicho nivel. Para ello trata la lista de Spawners de la sala como si estuviera ordenada teniendo en cuenta a que nivel pertenecen los jefes (es decir, la Abuela, al ser el jefe del primer nivel, tendría su Spawner en la primera posición de la lista).

Ahora bien, los nuevos jefes añadidos, a diferencia de la Abuela, tienen un comportamiento más complejo y necesitan conocer algunos datos sobre el escenario para su correcto funcionamiento. Por ello se tuvo que implementar una serie de componentes para configurar correctamente a dichos jefes antes de su generación Dichos componentes son el NerdConfigurator y el MotherSonController.

Al generador del Nerd se agregaría el llamado NerdConfigurator. Dicho componente lleva un registro ordenado de los NerdPoints de la sala del jefe y de los Spawners de aliados para el Nerd. Este componente pasa los NerdPoints al Nerd antes de que se genere y lo conecta con los Spawners de aliados.

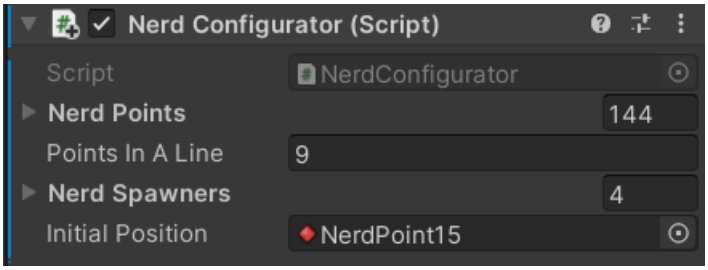


Figura 101: Componente NerdConfigurator

En cuanto a los generadores de la Madre e Hijo, estos llevarían el componente MotherSonConfigurator. Dicho componente lleva un registro de los puntos (PatrolPoints) entre los que se moverán la Madre y el Hijo en la sala (en el caso del Hijo, dichos puntos serán por los que se moverá cuando la Madre realice sus lanzamientos especiales). Este componente pasa al jefe dichos puntos.

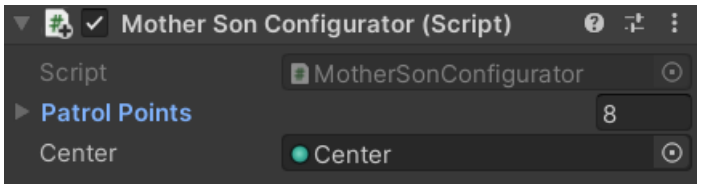


Figura 102: Componente MotherSonConfigurator

5.3.2. Control de retaguardia

Como ya se dijo anteriormente, para el movimiento de los enemigos de apoyo (sanadores y animadores) se decidió que estos trataran de mantenerse en todo momento detrás de los enemigos

atacantes respecto a la posición del jugador. Es decir, los enemigos de apoyo siempre se mantienen en la retaguardia de los atacantes.

Esto implica la necesidad de calcular en todo momento la posición de la retaguardia a partir de la posición del grupo de enemigos atacantes y del jugador. Se decidió que el responsable de esta tarea sería el HallController o controlador de la sala, agregándole una lista para los enemigos de ataque y otra para los de apoyo. Los enemigos son agregados a dichas listas durante su generación.

Para calcular la retaguardia, se decidió utilizar el siguiente algoritmo:

```
Si (atacantes.cuenta > 0 && apoyos.cuenta > 0){
        //Calculamos la posición del grupo de atacantes
        grupoAtaquePosición = (0,0)
        foreach (enemigo en atacantes){
                grupoAtaquePosición += enemigo.posición
        }
        grupoAtaquePosición = grupoAtaquePosición / atacantes.cuenta
        //Calculamos la posición de retaguardia
        distanciaJugadorAtacantes = Distancia(jugador.posición, grupoAtaquePosición)
        u = (grupoAtaquePosición – jugador.posición)/distanciaJugadorAtacantes
        posiciónRetaguardia = grupoAtaquePosición + retaguardiaDistancia * u.normalizado
        //Pasamos la retaguardia a los apoyos
        foreach(enemigo en apoyos){
                enemigo.ActivarSeguimientoDeRetaguardia()
                enemigo.PasarDatosRetaguardia(posiciónRetaguardia, radioÁreaRetaguardia)
        }
}
Sino (atacantes.cuenta == 0 && quedan enemigos){
        foreach(enemigo en apoyos){
                enemigo.DesactivarSeguimientoDeRetaguardia()
        }
}
```

El algoritmo anterior se ejecuta de forma continua durante el combate. La retaguardia se calcula a partir de la posición del jugador y de la posición del grupo de atacantes, para que en todo momento la retaguardia se encuentre detrás de los atacantes respecto al jugador.

En primer lugar, se obtiene la posición del grupo de enemigos de ataque calculando la media de las posiciones de los atacantes. Tras obtener dicha posición, a partir de ella y de la posición del jugador se calcula la posición de la retaguardia. Para dicho cálculo, también se precisa un dato más: la distancia establecida entre el grupo de atacantes y la retaguardia.

Tras obtener la posición de la retaguardia, dicha información se comunica a los enemigos de apoyo, junto con un parámetro que representa el radio del área de retaguardia. La retaguardia realmente es tratada como un área circular en la que se mantienen los enemigos de apoyo, en el que la posición de la retaguardia representa el centro de dicha área. Al pasar tanto la posición de la retaguardia y el radio del área de retaguardia a los apoyos, estos pueden saber cuándo pueden considerarse dentro de dicha área.

Estos cálculos solo se realizan si en la sala existen enemigos de ataque y enemigos de apoyo. Cuando los enemigos de ataque son derrotados al completo, si quedan enemigos de apoyo, el controlador de la sala desactivará el seguimiento de la retaguardia en los apoyo para que comiencen a huir.

Por último, habría que destacar que el controlador permite realizar ajustes a este algoritmo mediante la modificación de los parámetros RearAreaRatio y RearDistance, que representan respectivamente al radio del área de retaguardia y la distancia entre el grupo de atacantes y la retaguardia.

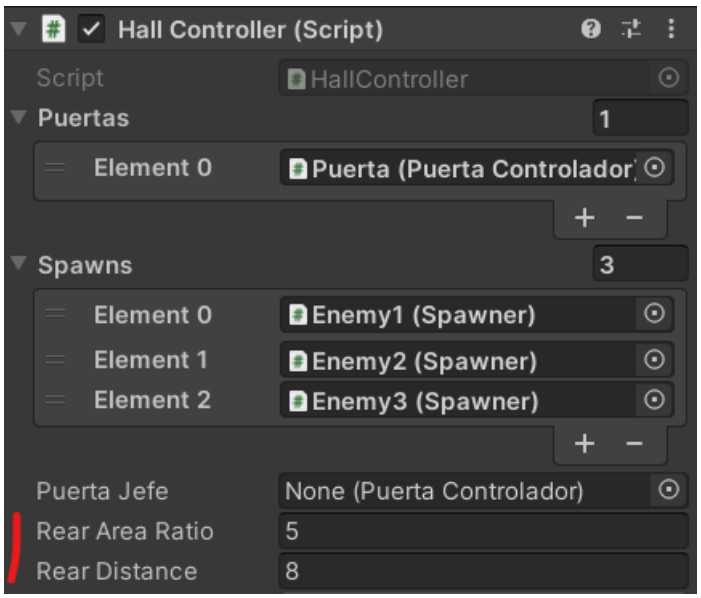


Figura 103: Parámetros de la retaguardia (RearAreaRatio y RearDistance)

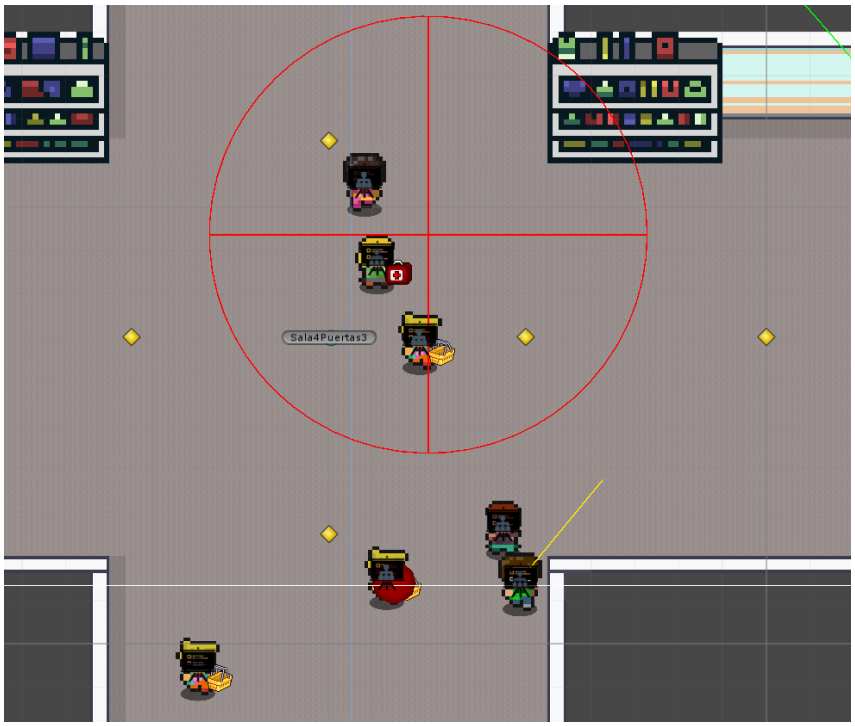


Figura 104: Ejemplo del movimiento en retaguardia

5.3.3. Muerte de los enemigos

Además de la generación de enemigos, también debemos tener en cuenta la muerte de estos y el efecto que esto conlleva. La muerte de los enemigos toma además una gran importancia en nuestro videojuego ya que la única forma de avanzar es derrotando a todos los enemigos que aparecen al encuentro del jugador.

Para el proyecto original, mi compañero Francisco Noguera ideó una manera para manejar la muerte de los enemigos. Todos los enemigos contaban con un componente llamado Enemigo (el cuál hemos renombrado en nuestro proyecto como Enemy) que tenía una referencia al componente SalaControlador (que hemos renombrado como HallController) de la sala en la que había sido generado el enemigo.

Al morir el enemigo, este accedía al controlador de la sala, para que el controlador borrase al enemigo de su registro y generar un Item en su posición.

Durante el desarrollo del proyecto, se realizaron algunas modificaciones sobre este sistema, destacando en primer lugar la relación del controlador de la sala con el enemigo tomando como ejemplo el patrón Observer, usando los Delegates y los Events de Unity. A los enemigos se les agregó un evento que se produce cuando estos mueren. Cuando el controlador genera los enemigos de la sala, se suscribe a los eventos de muerte de los enemigos. De esta forma evitamos que el enemigo acceda directamente al controlador.

Al morir el enemigo se produce su evento de muerte comunicando su derrota al controlador del enemigo. El controlador, al producirse el evento de muerte, ejecuta la rutina descrita anteriormente, eliminando del registro de enemigos al enemigo muerto, generando un Item en su posición, realizando después una comprobación de los enemigos que quedan en la sala.

En el siguiente diagrama se puede observar mejor como sucede la comunicación entre Enemy y HallController, mediante el evento OnDieEnemy. También se puede observar cómo, en el caso

de que se hayan derrotado a todos los enemigos y la sala fuera una sala de jefe (es decir, se venció al jefe de nivel), el HallController accede al controlador del nivel (NivelControlador) con NotifyEnemyDefeat para avisar al SceneController de que se ha derrotado al jefe. Para ello, NivelControlador produce el evento OnBossDefeated.

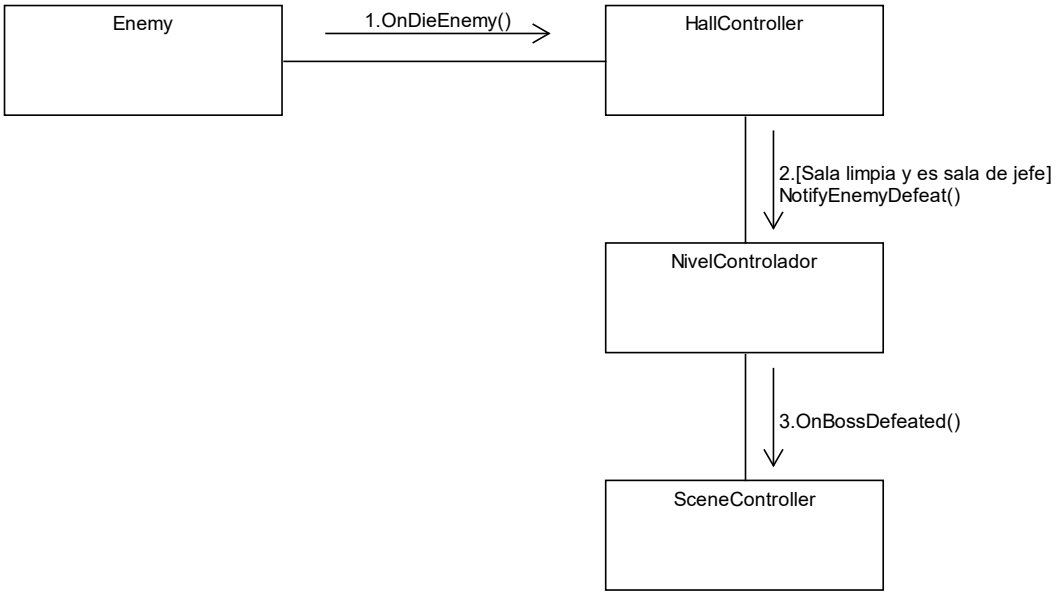


Figura 105: Muerte del enemigo

En el siguiente punto analizaremos con más detalle el SceneController y su relación con el controlador del nivel.

5.3.4. Control del nivel

Como ya se mencionó anteriormente, en el proyecto original solo se implementó un jefe de nivel: la Abuela. Lo que se traduce que en esencia solo se contaba con un único nivel. Cuando la Abuela era derrotada se reiniciaba todo el nivel (cargándose de nuevo la escena de Unity con el nivel) y se volvía a empezar.

Ahora, al tener tres jefes de nivel, la solución de simplemente reiniciar el nivel no nos sirve. Por ello se decidió crear la clase SceneController,componente responsable de cambiar de nivel o finalizar la partida, llevando para ello un registro de los niveles que componen el juego y cuál es el nivel en el que se encuentra el jugador en cada momento. El componente también se comunica con el controlador de la interfaz para actualizarla o mostrar mensajes en pantalla.

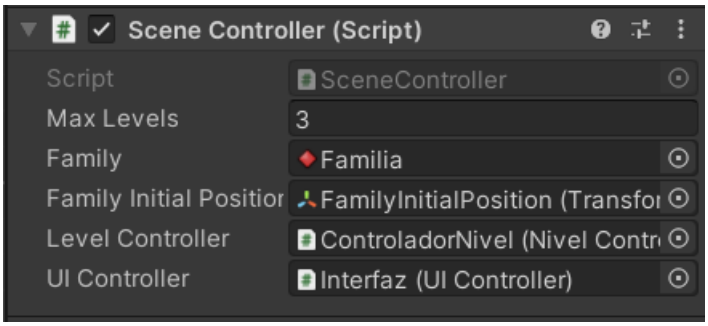


Figura 106: Componente SceneController

Para poder comunicar al SceneController cuando cambiar de nivel, se realizaron algunos cambios en el controlador del nivel (NivelControlador), aplicándose el patrón Observer, usando de nuevo Delegates y Events. Se creó un evento en el controlador del nivel para comunicar la muerte del jefe de nivel, evento al que se suscribe el SceneController.

Al producirse el evento, el SceneController muestra el mensaje de victoria en pantalla actualiza el registro del nivel actual y revisa su valor para determinar si ya se han completado todos los niveles o no. En el caso de que queden niveles pasados unos segundos se cambia el nivel. En el caso de que no quede ningún nivel por completar (se han completado todos) tras unos segundos, se finaliza el juego

Al cambiar de nivel se eliminan todos los accesorios del jugador y se carga de nuevo la escena con el nivel, para que, de esa forma, se vuelvan a reiniciar todas las salas y vuelvan a ensamblarse dichas salas con una nueva combinación (es decir, se genere un nuevo nivel). Tras cargar la escena, el SceneController inicializa el nivel, actualizando la interfaz, colocando al jugador en la sala inicial y conectando al SceneController con controlador del nuevo nivel.

Conectar el SceneController con el controlador del nivel se traduce en suscribir al SceneController al evento de muerte del jefe del controlador del nivel. También se actualiza la información referente al nivel actual y a la cantidad de niveles del controlador del nivel, algo indispensable, ya que el controlador de la sala del jefe consultará dicha información del controlador del nivel para saber que jefe generar.

Ahora bien, todo esto conlleva que al cargar la escena debe mantenerse en todo momento la misma instancia de SceneController. Por ello se decidió aplicar el patrón Singleton.

La finalidad del patrón Singleton es evitar que se cree más de una instancia de una determinada clase. Para ello se crea el objeto dentro de una clase y se recupera como una instancia estática [30].

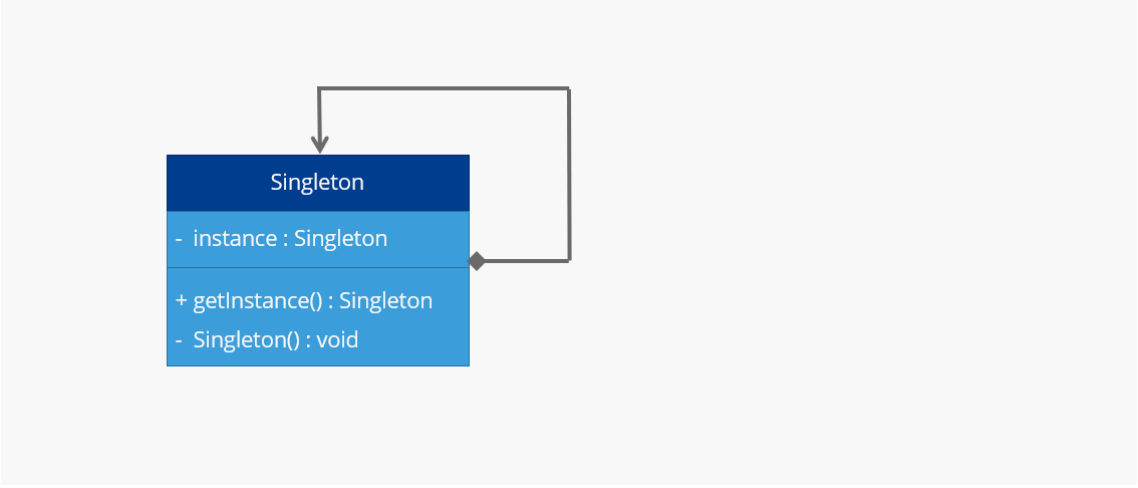


Figura 107: Esquema del patrón Singleton, IONOS Digital Guide

Al hacer que el SceneController herede de la clase Singleton, logramos que al cargar la escena no se pierda la instancia de SceneController y que no haya más de una instancia de dicha clase en ningún momento.

Por otra parte, además del SceneController también existen elementos que no queremos perder al cambiar de nivel, como es la familia (los personajes del jugador) ya que queremos mantener ciertas estadísticas del jugador entre los distintos niveles. Para ello, simplemente basta con hacer que dichos elementos sean hijos del GameObject con el script del SceneController.

Figura 108: Hijos del SceneController

De esta forma se consigue mantener entre nivel y nivel los personajes que han muerto del jugador, es decir, si en el primer nivel muere el Padre, al iniciarse el siguiente nivel el jugador no tendrá dicho personaje. Por desgracia no se consiguió mantener las estadísticas de vida y energía entre los niveles, que se restauran cada vez que el jugador pasa de nivel. En cuanto a los accesorios equipados por el jugador, aunque se consiguió encontrar una manera de mantenerlos entre los niveles haciendo que se convirtieran en hijos de la Familia, hubo ciertos problemas relacionados con sus efectos, ya que dejaban las estadísticas de los personajes con valores incorrectos, por lo que finalmente se decidió que lo mejor era deshacerse de ellos al finalizar el nivel.

En lo que respecta a la finalización del juego, el SceneController simplemente carga la escena de Unity con la pantalla de inicio. Al cargarse la pantalla de inicio, el SceneController se autodestruye con sus hijos, para evitar así que se mantenga el componente en más escenas después de haberse terminado el juego.

Con todo esto se cumpliría la UT de Cambio de nivel al matar al jefe de nivel.

Un último detalle para destacar, aunque no tenga mucha relación con los enemigos, es que también se ha hecho que el SceneController sea el responsable de finalizar el juego cuando mueren todos los personajes del jugador.

5.4. Metodología

Para el desarrollo de este proyecto se decidió seguir una metodología ágil basada principalmente en la metodología Scrum, usando Jira como herramienta de apoyo. En dicha herramienta se creó una tabla, siguiendo el ejemplo de las tablas Kanban, para organizar el desarrollo de las distintas tareas del proyecto. Se organizaron cuatro columnas, representando las distintas fases en las que se pueden encontrar las tareas. Dichas columnas son las siguientes: “Por hacer” (la columna de las tareas que aún no se han iniciado), “Haciendo” (la columna de las tareas que están actualmente en desarrollo), “Revisión” (la columna de las tareas que son revisadas antes de su finalización” y “Finalizado” (la columna de las tareas que se han finalizado).

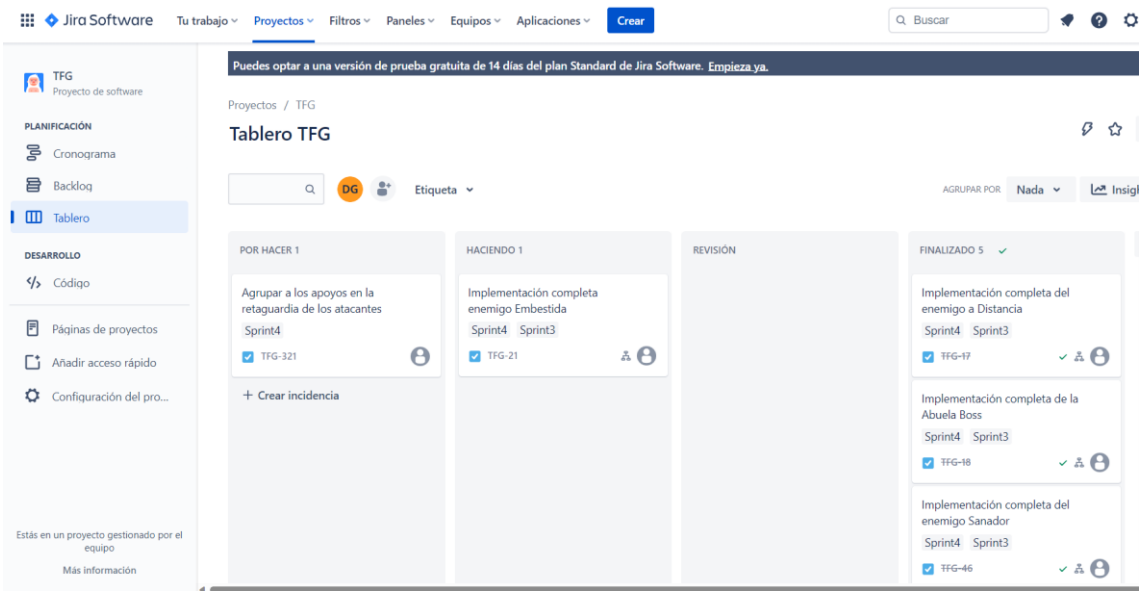


Figura 109: Tabla de Jira

Como ya se dijo en puntos anteriores, se decidió encapsular las tareas o elementos específicos relacionados con el desarrollo del videjuego en UTs. Dichas UTs se han representado en Jira mediante las Incidencias, estructuras de información de la aplicación para representar las tareas de un proyecto. En cuanto a las PA de las UT, se decidió representarlas dentro de las UT como subtareas (Incidencias dentro de una Incidencia) al no contar en Jira con un elemento específico semejante a las PA.

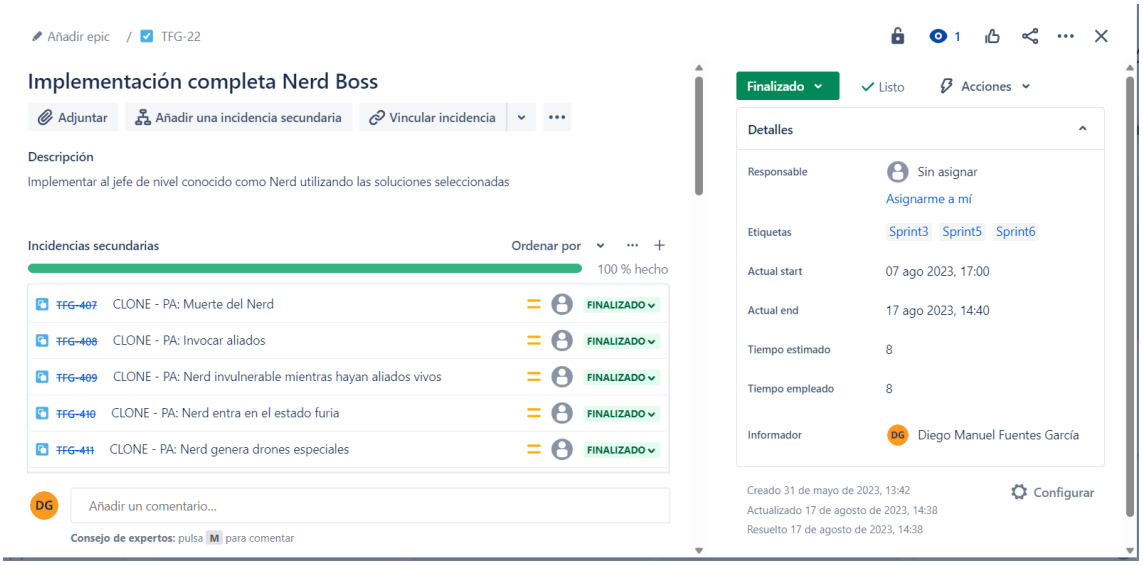
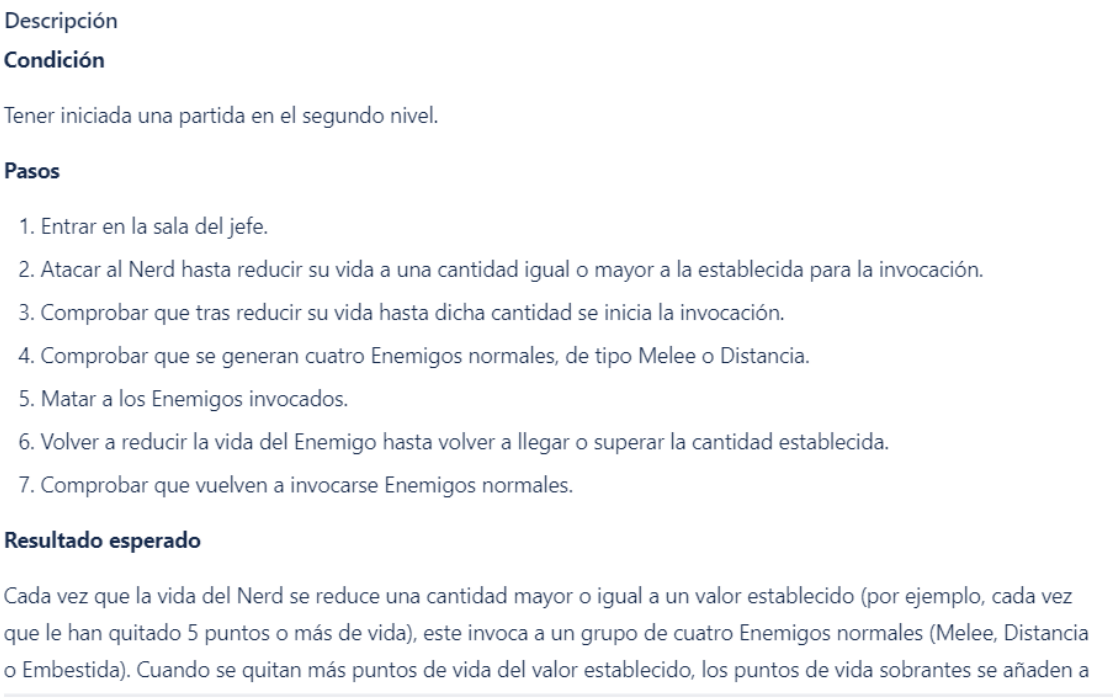


Figura 110: Incidencia de Jira

Para describir cada PA se decidió seguir la plantilla Condición-Pasos-Resultado esperado, con la que se describe la condición para poder realizar la PA, los pasos a seguir y el resultado esperado tras seguir dichos pasos.


Descripción

Condición

Tener iniciada una partida en el segundo nivel.

Pasos

1. Entrar en la sala del jefe.
2. Atacar al Nerd hasta reducir su vida a una cantidad igual o mayor a la establecida para la invocación.
3. Comprobar que tras reducir su vida hasta dicha cantidad se inicia la invocación.
4. Comprobar que se generan cuatro Enemigos normales, de tipo Melee o Distancia.
5. Matar a los Enemigos invocados.
6. Volver a reducir la vida del Enemigo hasta volver a llegar o superar la cantidad establecida.
7. Comprobar que vuelven a invocarse Enemigos normales.

Resultado esperado

Cada vez que la vida del Nerd se reduce una cantidad mayor o igual a un valor establecido (por ejemplo, cada vez que le han quitado 5 puntos o más de vida), este invoca a un grupo de cuatro Enemigos normales (Melee, Distancia o Embestida). Cuando se quitan más puntos de vida del valor establecido, los puntos de vida sobrantes se añaden a

Figura 111: Ejemplo de descripción de una PA

Como ya se mencionó en el plan de trabajo, el desarrollo se dividió en una serie de sprints de dos semanas. El desarrollo normal de cada sprint podría encapsularse en las siguientes fases:

- Inicio del Sprint. Teniendo en cuenta el desarrollo del anterior Sprint anterior y la capacidad de trabajo, se establecen las UTs que se desarrollarán en el nuevo sprint, colocándose en la columna “Por hacer” de Jira. Dichas UTs se obtienen del Backlog, de Sprints anteriores en el caso de que no se haya completado una UT necesaria o son creadas de cero. La razón de crear una nueva UT puede ser para continuar alguna parte de otra UT que no se finalizó en Sprints anteriores o por detectar un nuevo requisito, tarea, etc. necesaria para el desarrollo del proyecto.
- Desarrollo del Sprint. Se van desarrollando las tareas relacionadas con las UTs, moviéndose sus tarjetas a la columna “Haciendo”. Cuando una UT se considera completada, se mueve su tarjeta a la columna “Revisión”, comprobándose uno a uno el cumplimiento de las PA de la UT. En el caso de que no se cumpla alguna, se regresa a la columna “Haciendo” para solucionar el problema, pero si todas se cumplen, se considera finalizada la UT y se mueve a la columna “Finalizadas”.
- Fin del Sprint. Se realiza una reunión con el tutor para revisar el desarrollo del proyecto.

En esencia, se pueden distinguir un total de seis sprints, siete si consideramos que existió un Sprint 0 durante el que se estudiaron las técnicas de IA utilizadas en los videojuegos, entre el 16 y el 30 de mayo, hasta escoger los BT. Dichos Sprints son los siguientes:

- Sprint 1 (30 de mayo-13 de junio). Durante este Sprint se estudiaron las distintas soluciones disponibles para el uso de BTs y para conseguir el movimiento de los enemigos. El principal objetivo de este Sprint fue seleccionar las librerías necesarias para la implementación de la IA. Al finalizar el Sprint, se seleccionaron los paquetes MonoBehavourTree y NavMeshPlus.

- Sprint 2 (13 de junio-27 de junio). En este Sprint se inició el verdadero desarrollo del proyecto, comenzando a implementarse las versiones Sandbox de los enemigos. El objetivo de este sprint fue completar dichas versiones de los enemigos, sin llegar a integrarlas en el juego. No se llegó a completar el objetivo, pero se llegaron a implementar la mayoría de los enemigos básicos y el primer jefe de nivel. Este Sprint sirvió además para medir mi capacidad de trabajo para el proyecto, estimándose entre 30 y 40 horas.
- Sprint 3 (27 de junio – 11 de julio). En este Sprint se marcó como objetivo terminar las implementaciones Sandbox de los enemigos e integrarlos en el juego. Dicho objetivo no se completó, pero se consiguió integrar la mayoría de enemigo básicos en el juego y completarse la versión Sanbox del jefe del segundo nivel (el Nerd).
- Sprint 4 (11 de julio – 25 de julio). Para este Sprint se marcó como objetivo principal preparar las bases de la memoria, sin dejar de trabajar en el proyecto, siguiendo con la integración del resto los enemigos e investigando el movimiento en retaguardia para los apoyos. Como resultado se completaron las bases de varios puntos importantes de esta memoria, además de terminar por completo la integración de la mayoría de los enemigos básicos y del primer jefe de nivel (la abuela).
- Sprint 5 (25 de julio – 8 de agosto). Durante este Sprint se marcó como objetivo completar e integrar el enemigo con embestida, el jefe de segundo nivel y el del tercero, además de ir incluyendo animaciones y efectos para los enemigos. Como resultado, se integró el enemigo con embestida y se completó la versión Sandbox de la madre y el hijo (jefes del tercer nivel).
- Sprint 6 (8 de agosto – 22 de agosto). Para este Sprint se marcó como objetivo integrar los jefes del segundo nivel y del tercero, completar los efectos y animaciones, además de subir el juego terminado y realizar una encuesta para recoger la opinión de los usuarios respecto al juego. Como resultado, se terminó el videojuego, se subió a Itch.io y se preparó una encuesta de Google.

Tras todo ello, las últimas semanas antes del 7 de septiembre, fecha límite para la entrega del TFG, se centraron los esfuerzos en completar la memoria del proyecto.

6. Implantación y pruebas

Tras finalizar el desarrollo, obteniéndose una versión funcional y jugable de Black Friday War, se decidió publicar el videojuego en Itch.io. Dicha plataforma ya fue utilizada para la adquisición de recursos artísticos para el videojuego, pero en realidad es una plataforma de distribución de videojuegos independientes, aunque su catálogo también cuenta con distintos packs, plugins y elementos que pueden servir en el desarrollo de videojuegos.

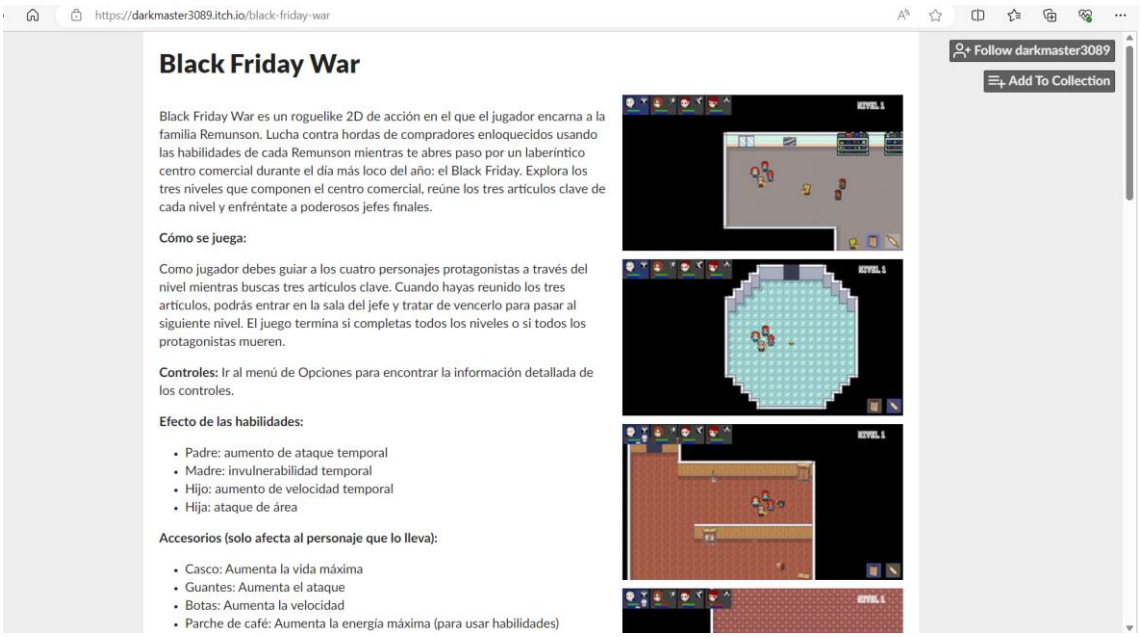


Figura 112: Página de Black Friday War en Itch.io

También se decidió crear un formulario de Google para conocer la opinión de los usuarios sobre el juego y los enemigos. Para obtener el mayor número de datos se decidió distribuir el juego entre algunos conocidos, pidiendo a estos a su vez que ofrecieran probar el juego a sus conocidos.

En total se consiguieron setenta y siete visitas en la página de Black Friday War y doce descargas del videojuego. De entre los doce usuarios, cuatro respondieron a la encuesta. Aunque no hayamos conseguido muchas respuestas en la encuesta, las que hemos obtenido nos han permitido ofrecer un poco de luz sobre el rendimiento del juego entre los jugadores. A continuación, analizaremos las respuestas.

En la primera pregunta, referente a lo lejos que han llegado los jugadores, podemos observar que solo la mitad ha llegado al segundo nivel y la otra mitad se quedó en el primer nivel. Ninguno parece haber sido capaz de superar el juego o de, al menos, llegar al tercer nivel.

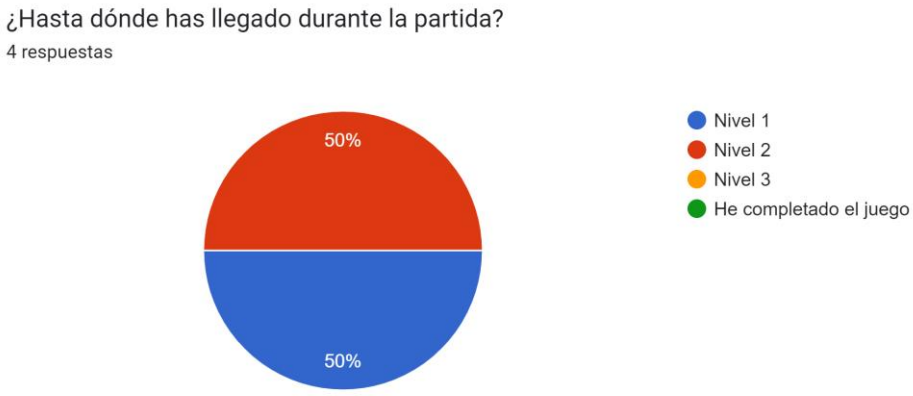


Figura 113: Resultados de la primera pregunta

En la segunda pregunta, referente a la dificultad de los combates contra los enemigos normales, el 75% han definido los combates como normales (ni fáciles ni difíciles), mientras que un 25% los ha definido como muy fáciles.

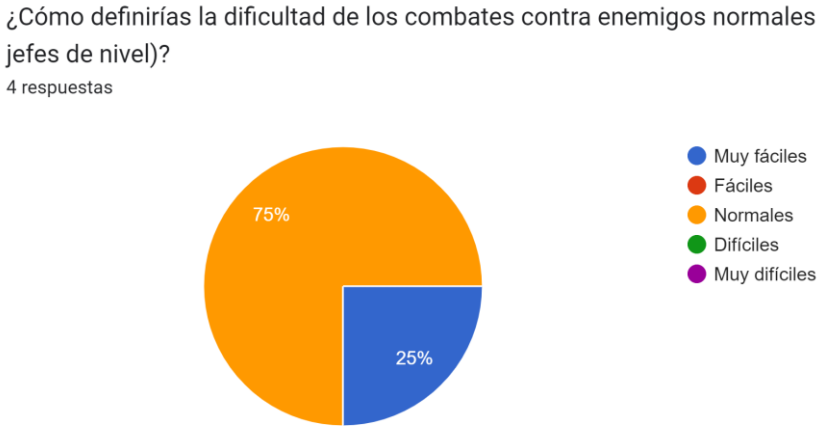


Figura 114: Resultados de la segunda pregunta

En la tercera pregunta, la mitad de los usuarios ha definido como “mucha” la cantidad de enemigos generados en las salas mientras que la otra mitad la ha definido como una cantidad normal.

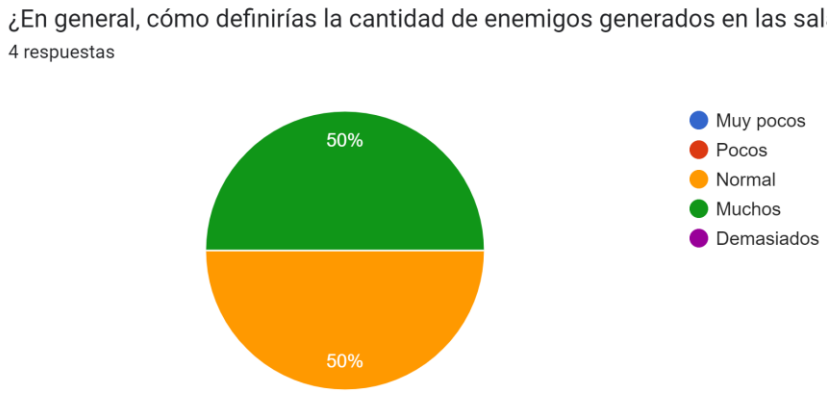


Figura 115: Resultados de la tercera pregunta

En cuanto a la vida de los enemigos, en la cuarta pregunta el 75 % de los usuarios la definieron como una cantidad adecuada (normal) mientras que un 25% la definió como demasiado grande.

Figura 116: Resultados de la cuarta pregunta

Respecto a la velocidad de los enemigos, en la quinta pregunta el 75% usuarios la definieron como una velocidad adecuada (normal) mientras que un 25% definió a los enemigos como rápidos.

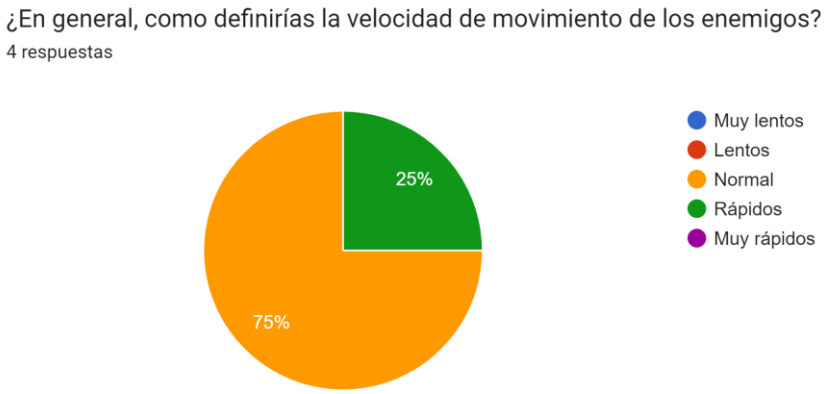


Figura 117: Resultados de la quinta pregunta

En la sexta pregunta, quedó claro que en general se puede distinguir la división de los enemigos en atacantes y apoyos, ya que el 75% de los usuarios reconoció dicha división.

Figura 118: Resultados de la sexta pregunta

En lo que respecta a la formación de los apoyos en la retaguardia de los atacantes, todos los usuarios estuvieron de acuerdo en que era buena idea que los apoyos se mantuvieran detrás de los atacantes.

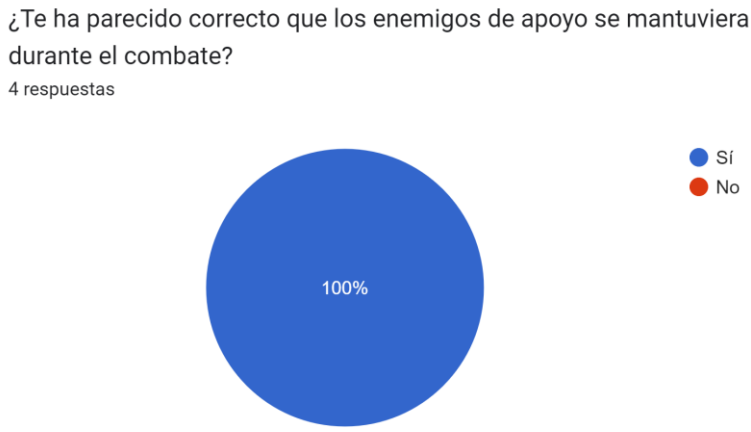


Figura 119: Resultados de la séptima pregunta

Todos los usuarios estuvieron de acuerdo también en que la presencia de los enemigos de apoyo agregó dificultad a los combates.

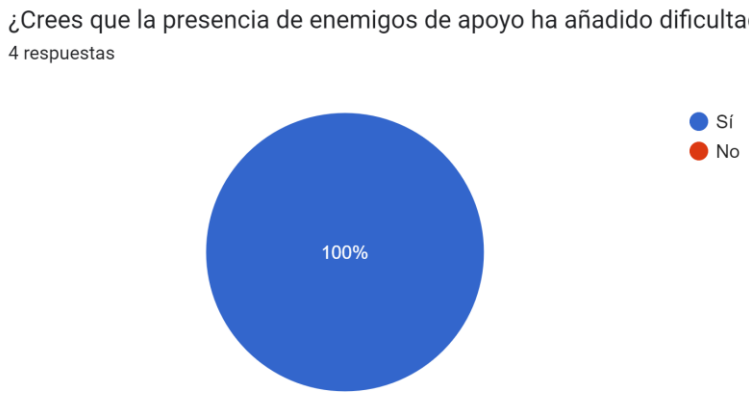


Figura 120: Resultados de la octava pregunta

También podemos decir que el sanador y su función es identificable por el usuario, ya que el 75% notó durante la partida la existencia del sanador.

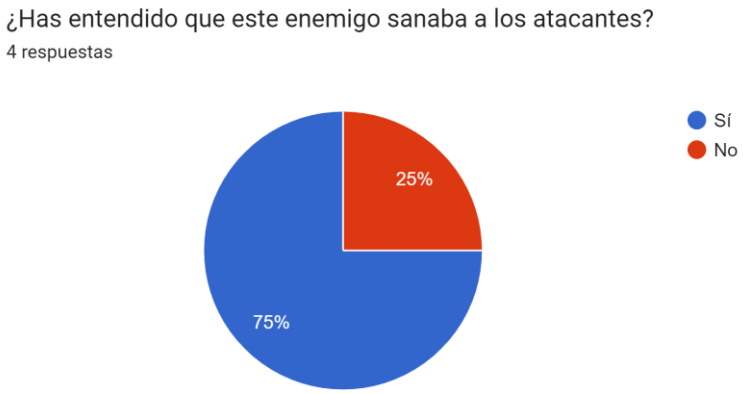


Figura 121: Resultados de la novena pregunta

El apoyo cuya función no parece haber quedado del todo clara ha sido la del animador, ya que solo el 50 % reconoció la presencia de este enemigo y su capacidad para potenciar a los atacantes.

Figura 122: Resultados de la décima pregunta

Respecto a la potenciación aplicada por el animador, todos los usuarios la definieron como correcta y adecuada.

Figura 123: Resultados de la undécima pregunta

En cuanto a la duración de la potenciación, el 75% la definió como correcta mientras que un 25% la definió como larga.

Figura 124: Resultados de la duodécima pregunta

Respecto al ataque con embestida, parece que ha tenido buena acogida, ya que, aunque el 50% de los usuarios les ha parecido normal, al otro 50% les ha parecido una mecánica interesante.

Figura 125: Resultados de la décimo tercera pregunta

En cuanto a la dificultad de los jefes de nivel, el 50% de los jugadores los encontraron difíciles y el otro 50% los encontraron normales y adecuados.

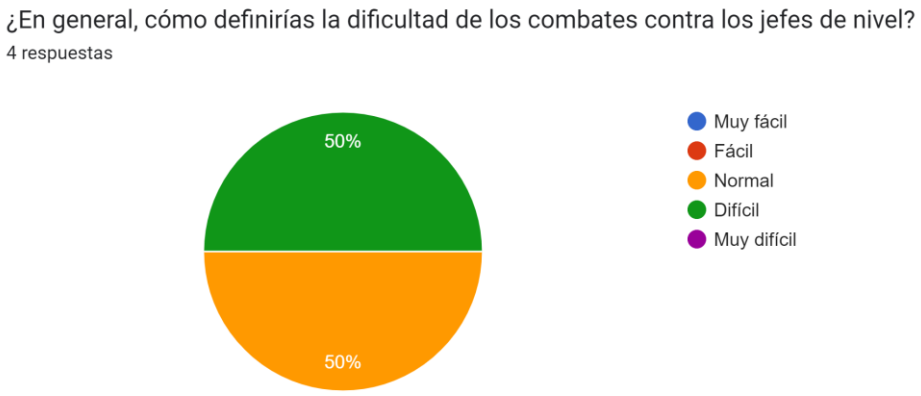


Figura 126: Resultados de la décimo cuarta pregunta

Respecto al jefe del primer nivel, el 50% de los jugadores lo encontró difícil, el 25% lo encontró normal y el 25% lo definió como un jefe muy fácil.

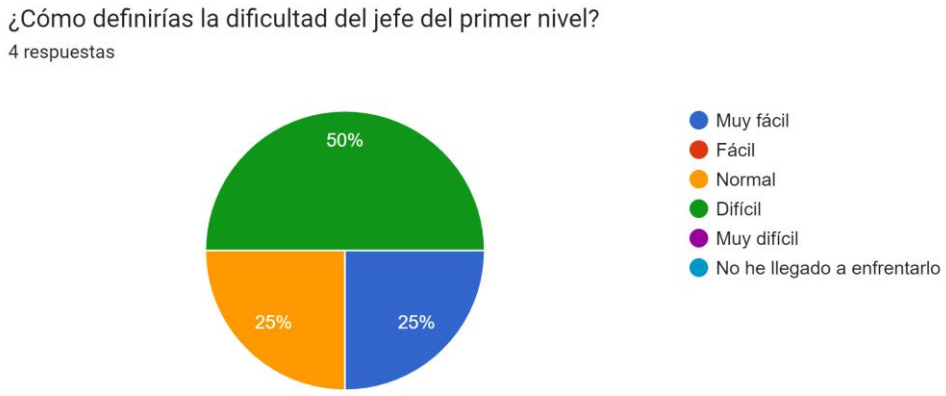


Figura 127: Resultados de la décimo quinta pregunta

En cuanto a sus ataques, el 75% los definió como normales y un 25 % los encontró interesantes.

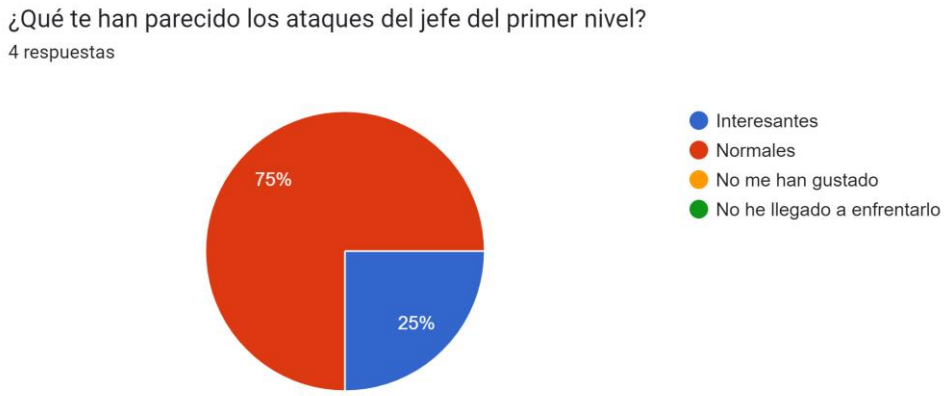


Figura 128: Resultados de la décimo sexta pregunta

Respecto a la dificultad del jefe del segundo nivel, el 50% de los usuarios (los que se quedaron en el primer nivel) no llegaron a enfrentarlo y el 50% que llegó a enfrentarlo lo definieron como un jefe difícil.

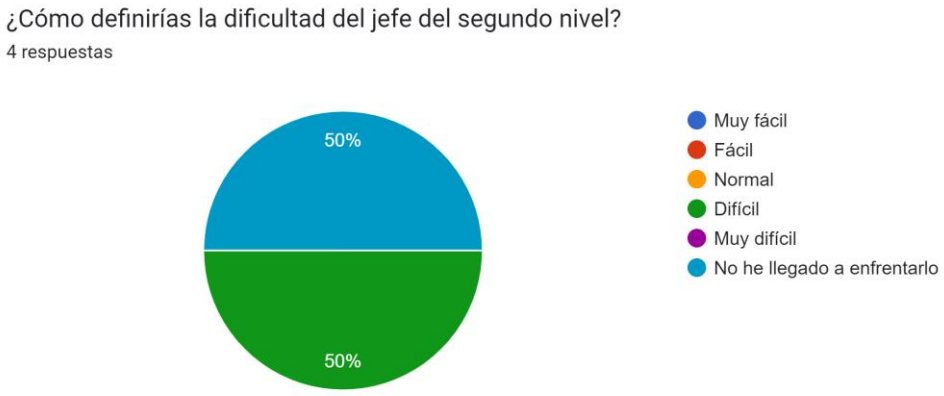


Figura 129: Resultados de la décimo séptima pregunta

En cuanto a sus ataques, la mitad de los que lo enfrentaron los definieron como normales mientras que la otra mitad los definió como interesantes.

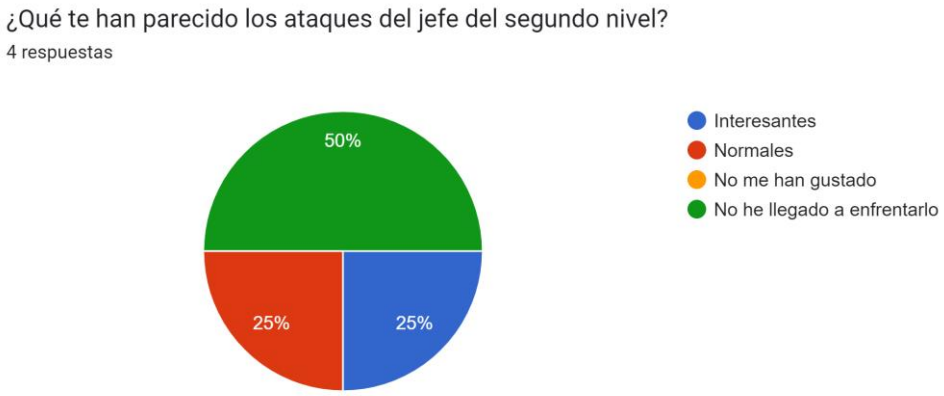


Figura 130: Resultados de la décimo octava pregunta

Respecto a la presencia de los drones especiales durante el combate contra este jefe, todos los jugadores que lo enfrentaron estuvieron de acuerdo en que llegaron a hacer que el combate fuera frustrante.

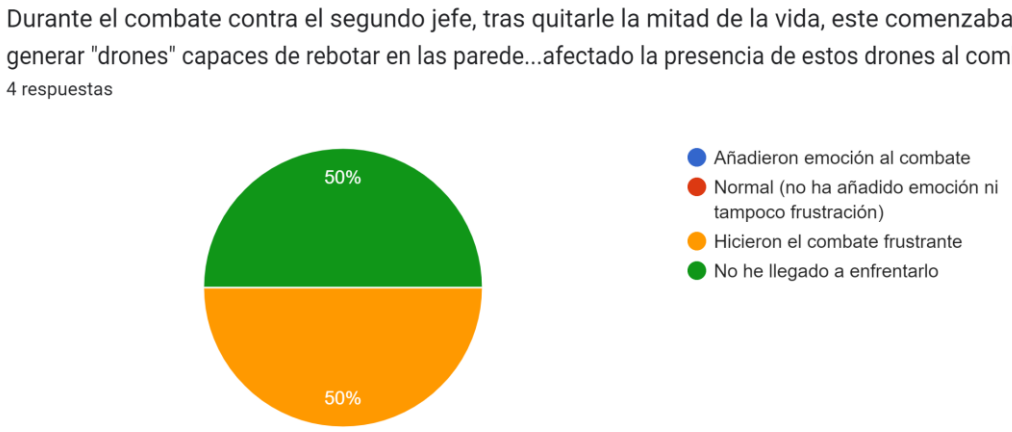
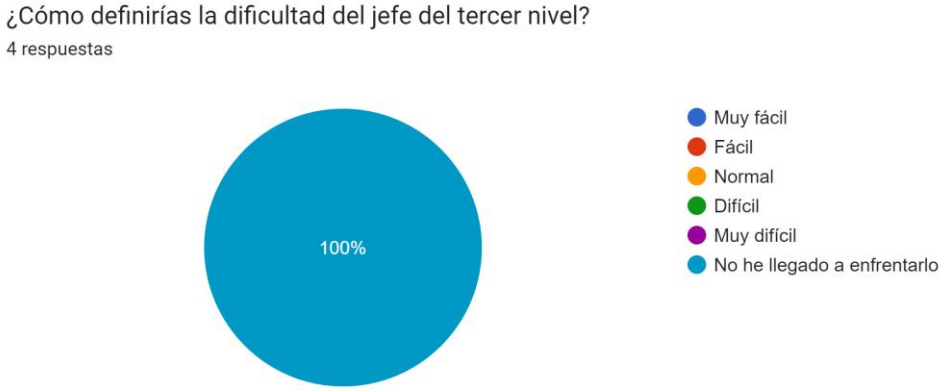
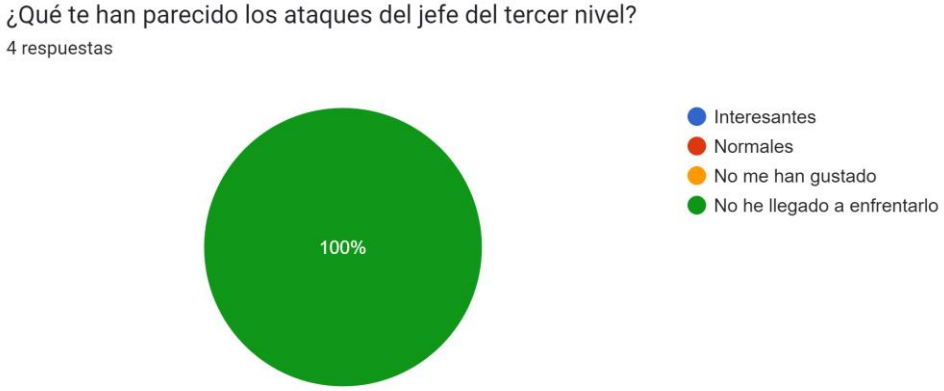


Figura 131: Resultados de la décimo novena pregunta

No podemos sacar conclusiones respecto al tercer jefe, ya que ningún jugador llegó a enfrentarse con él, aunque se puede deducir por las respuestas anteriores, que lo más seguro es que lo hubieran encontrado difícil e incluso frustrante por culpa de las pelotas lanzadas por el jefe.





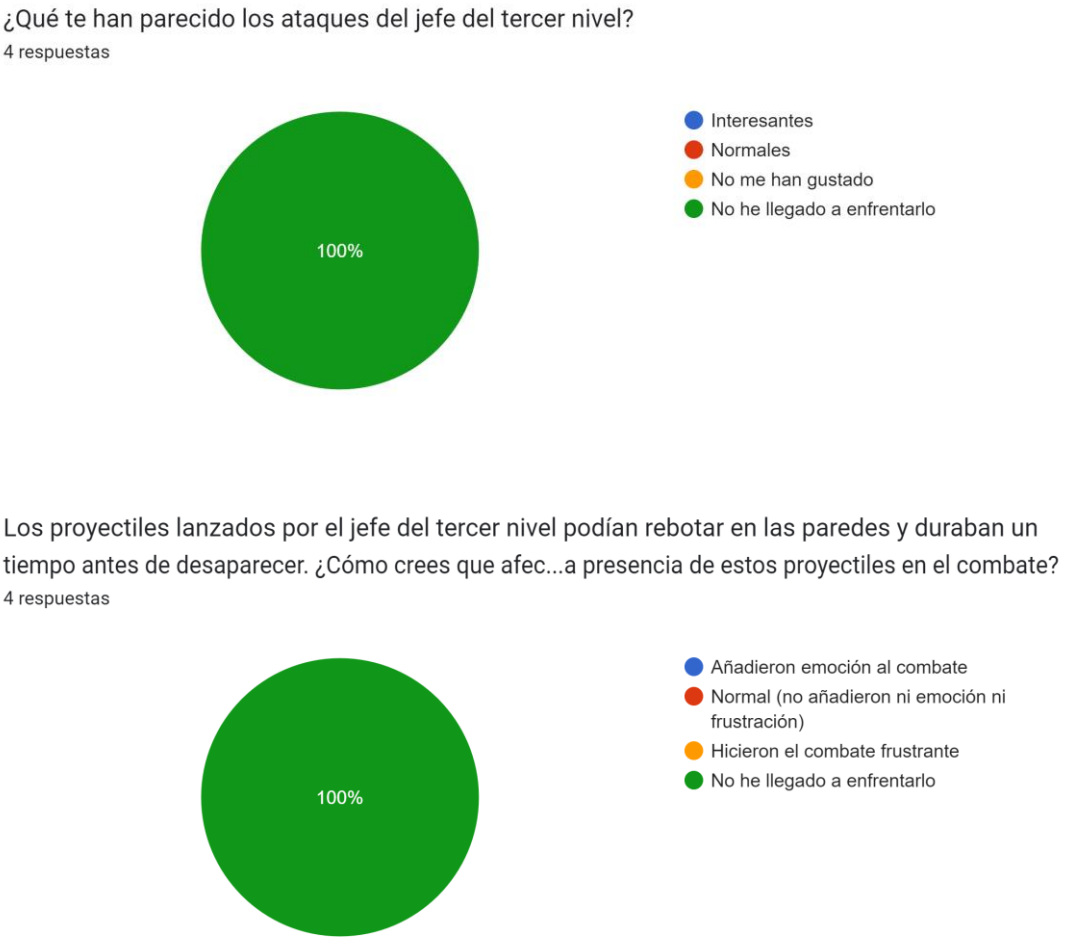


Figura 132: Resultados de las preguntas sobre el tercer jefe

Respecto a la pregunta sobre qué enemigo normal gustó más, la mayoría de los usuarios tuvieron predilección por el enemigo con embestida, mientras que el resto votó por el animador y el enemigo a melee.

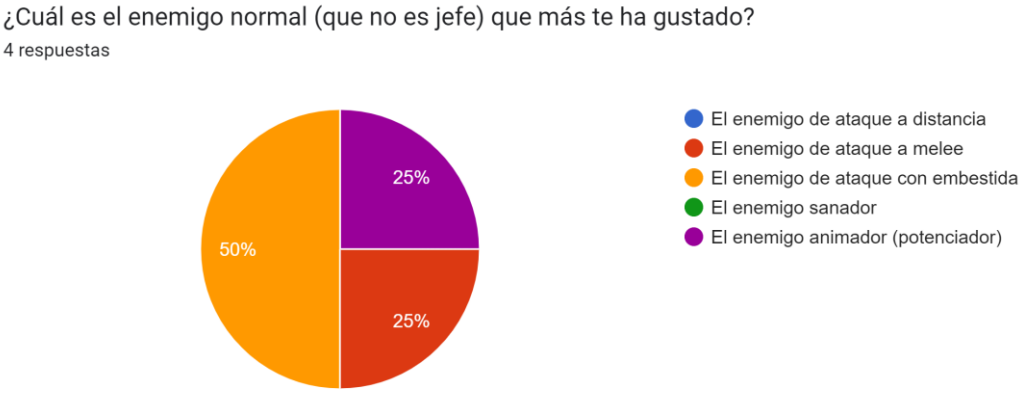


Figura 133: Valoración de los enemigos normales

En cuanto a la popularidad de los jefes, el jefe del primer nivel se lleva el 75% de las votaciones mientras que el resto votó por el del segundo jefe. No es de extrañar este resultado, ya que el 50% únicamente se enfrentó al jefe del primer nivel. En cuanto a ese 25% que enfrentó al segundo jefe y votó al primero, lo más seguro es que se decantara por el primero debido a la dificultad del combate contra el segundo jefe.

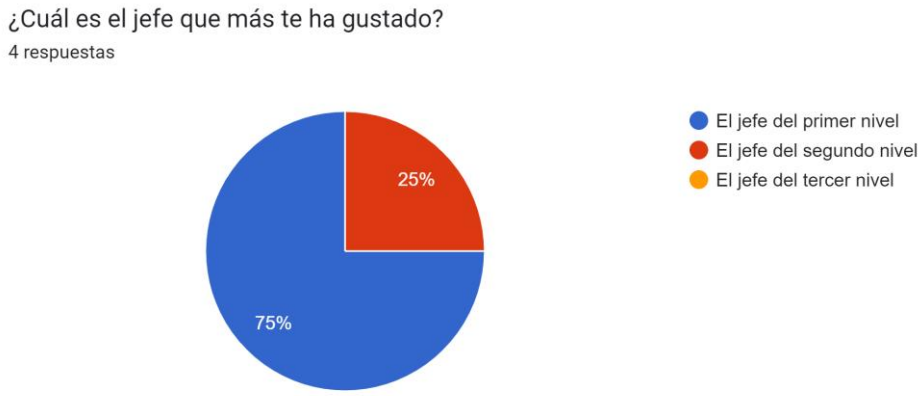


Figura 134: Valoración de los jefes

Respecto a la valoración del juego, el 25% le dio una nota de 7 y el 75% una nota de 8. Podemos deducir por tanto que el juego ha llegado a gustarles a los usuarios a pesar de no haberlo encontrado perfecto.

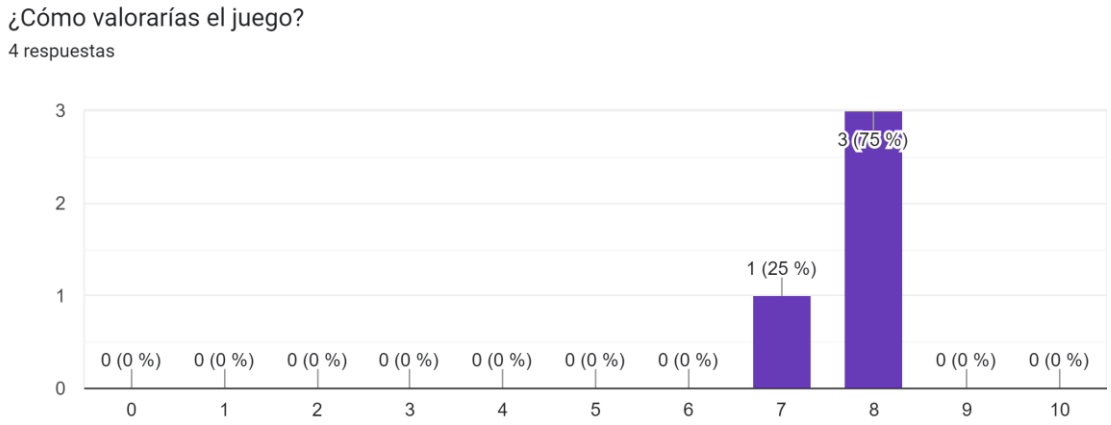


Figura 135: Valoración del juego

En última instancia, tenemos las preguntas de respuesta, que se añadieron en la encuesta para conocer más profundamente la opinión de los usuarios respecto a los enemigos y al juego en general.

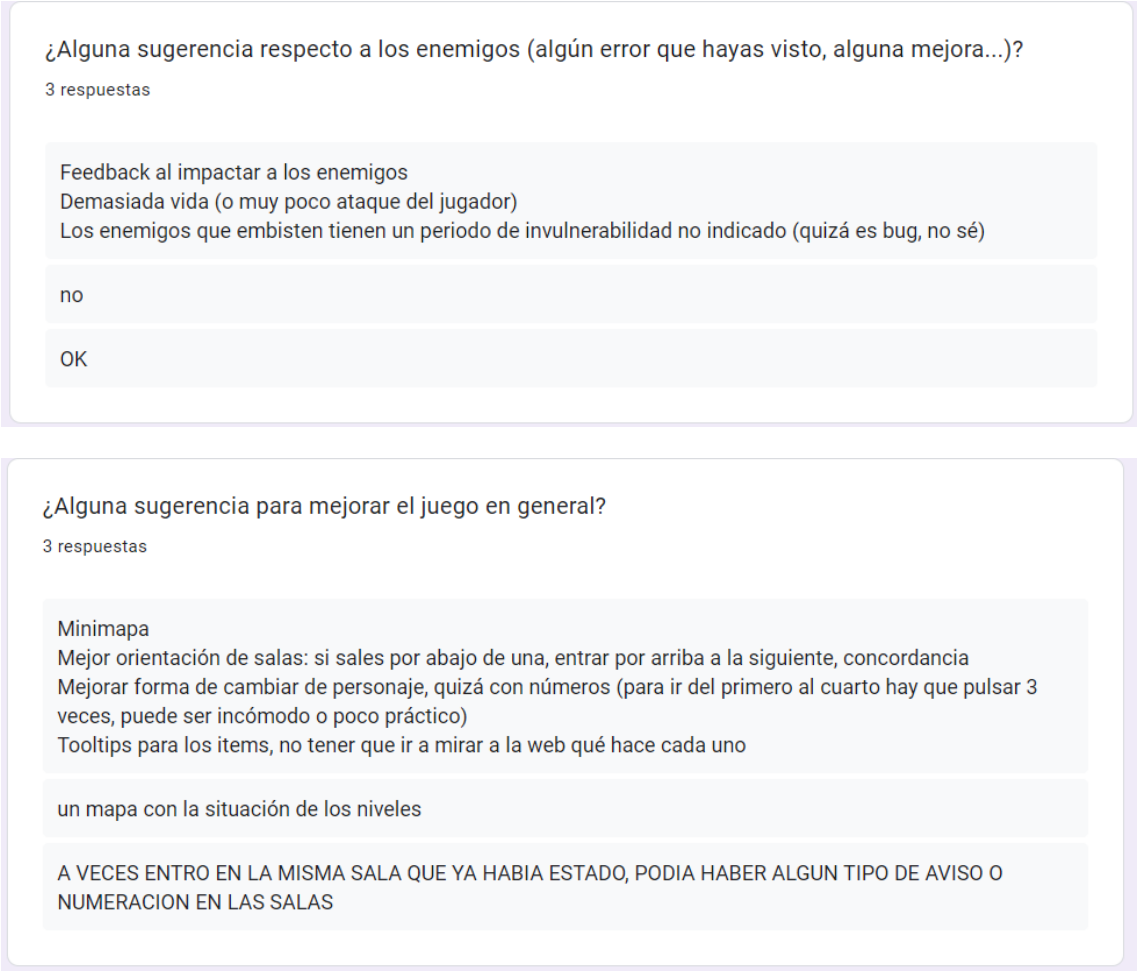


Figura 136: Sugerencias y anotaciones de los usuarios

Tras analizar estos resultados podemos deducir la adición en los combates de los enemigos apoyo y del enemigo con embestida parece haber sido un acierto, consiguiéndose unos combates más emocionantes y entretenidos. Por otra parte, parece ser necesaria una revisión en la dificultad de los enemigos, ya que algunos jugadores han encontrado muy fáciles o difíciles algunos enemigos, siendo incapaces de alcanzar el tercer nivel.

A pesar de ello, podemos determinar que el juego, en general, ha tenido buena acogida entre los usuarios que lo han jugado.

7. Conclusiones y trabajos futuros

7.1. Conclusiones

Tras finalizar el proyecto podemos concluir que la mayoría de los objetivos planteados para este TFG se han completado.

En primer lugar, podemos observar que todos los enemigos que se idearon para el proyecto han sido completado e integrados de forma satisfactoria en el videojuego. Es cierto que, debido a la falta de tiempo y a la complejidad de algunas ideas, algunas mecánicas de los enemigos no han llegado a implementarse, como es el caso de los ataques con comba del jefe del tercer nivel, pero a pesar de ello se ha logrado un videojuego funcional y jugable que ha llegado a publicarse en una plataforma de distribución de videojuegos.

Gran parte de que el proyecto se haya desarrollado de forma exitosa se lo debemos a las metodologías ágiles que se han seguido a lo largo de estos meses de trabajo.

También, gracias al estudio y uso de técnicas de IA para la implementación de los patrones de comportamiento de los enemigos, se ha conseguido aprender más sobre este fascinante campo, especialmente en lo que respecta a su aplicación en los videojuegos.

Lo único que no ha llegado a realizarse es la idea de incluir una IA táctica para coordinar los movimientos de los enemigos y realizar tácticas. Podríamos considerar que el movimiento de los enemigos de apoyo en la retaguardia es una especie de táctica, pero realmente no se ha llegado a crear una componente de IA que administre y coordine a los enemigos.

En esencia, a pesar de no haber implementado una IA táctia, podemos concluir que el TFG se ha completado de forma satisfactoria.

7.2. Relación con los estudios cursados

En cuanto a la relación de este TFG con los estudios cursados, en primer lugar, se deberían destacar las asignaturas de Ingeniería del software (11555), Proceso del software (11571) y Proyecto de ingeniería de software (11574) a la hora de llevar a cabo este TFG como un proyecto software seleccionando y siguiendo metodologías ágiles.

También debemos destacar la importancia de la asignatura de Diseño de Software (11565) a la hora de seleccionar y aplicar los patrones Singleton y Observer en nuestro proyecto. Deberíamos mencionar además la asignatura de Mantenimiento y evolución del software (11569), ya que, siguiendo lo aprendido en ella, se decidió tratar seguir la guía de estilo del lenguaje C# a la hora de implementar y documentar los distintos scripts que se han desarrollado a lo largo del proyecto.

Por último, debemos hacer mención especial a las asignaturas de Desarrollo de Videojuegos 3D (11645) y Desarrollo de Videojuegos 2D (14101), ya que en dichas asignaturas se presentó el desarrollo de videojuegos y el motor Unity. Sin los conocimientos brindados por estas asignaturas, el desarrollo de este proyecto hubiera sido mucho más duro e incluso tal vez ni siquiera se hubiera llegado a desarrollar este proyecto (recordemos que este TFG deriva de un proyecto de la asignatura de Desarrollo de Videojuegos 2D).

7.3. Trabajos futuros

Como trabajos futuros, una posibilidad sería la de desarrollar una IA táctica que administre a los enemigos para que coordinen sus ataques y realicen distintas tácticas durante el combate, evitando así que los atacantes tiendan a concentrarse en los mismos puntos. También podría revisarse la dificultad de los enemigos e intentar fortalecer a los enemigos normales a medida que el jugador avanza por los niveles. También podrían añadirse nuevos tipos de enemigos o crearse diversas variantes para cada tipo de enemigo, algo que se llegó a plantear en el GDD de Black Friday War.

Otra posibilidad sería la de crear nuevas armas para el jugador, el cuál de momento solo cuenta con dos. También podría revisarse el arte de los niveles y buscarse nuevas plantillas para que cada nivel tenga una ambientación distinta que los diferencie entre ellos (podría seguirse la idea del GDD de que cada nivel represente una sección diferente del centro comercial).

También sería buena idea tener en cuenta algunas de las sugerencias que los usuarios nos comunicaron a través de la encuesta, como la de añadir un mapa para evitar que los jugadores se pierdan, revisar la vida de los enemigos, agregar indicaciones de los efectos de los accesorios del juego y más retroalimentación referente a los estados de los enemigos (especialmente respecto al período de invulnerabilidad del enemigo con embestida).

Bibliografía

[1] Definición de videojuego – Ceibal formación. Consultado en https://blogs.ceibal.edu.uy/formacion/faqs/que-es-un-videojuego/

[2] Historia de los videojuegos – Retro Informática. Consultado en https://www.fib.upc.edu/retro-informatica/historia/videojocs.html

[3] Análisis del impacto del COVID-19 y mejores prácticas de teletrabajo en el sector de los videojuegos – IDG CONSULTING. Consultado en www.aevi.org.es/web/wp-content/uploads/2021/07/Análisis-del-impacto-del-COVID-19-y-mejores-prácticas-de-teletrabajo-en-el-sector-de-los-videojuegos.pdf

[4] Alex Pareja *La emocionante evolución del Roguelike*. Consultado en La emocionante evolución del Roguelike: orígenes, nicho y su llegada a la primera plana en videojuegos como Deathloop, Hades, Returnal... (ign.com)

[5] Berlin Interpretation. Consultado en Berlin Interpretation - RogueBasin

[6] Rogue – Wikipedia. Consultado en https://es.wikipedia.org/wiki/Rogue

[7] Álvaro Corazón Rural *Gauntlet, el sacaperras del guerrero*. Consultado en https://www.jotdown.es/2018/08/gauntlet-el-sacaperras-del-guerrero/

[8] Gauntlet – Wikipedia. Consultado en https://es.wikipedia.org/wiki/Gauntlet_(videojuego)

[9] The Binding of Isaac – Wikipedia. Consultado en https://es.wikipedia.org/wiki/The_Binding_of_Isaac

[10] John Teti *The Binding of Isaac*. Consultado en https://www.eurogamer.net/the-binding-of-isaac-review

[11] Enter the Gungeon – Wikipedia. Consultado en https://es.wikipedia.org/wiki/Enter_the_Gungeon

[12] Luis Vazquez *Análisis de Enter the Gungeon.* Consultado en https://vandal.elespanol.com/analisis/ps4/enter-the-gungeon/27380#p-73

[13] Hades – Wikipedia. Consultado en https://es.wikipedia.org/wiki/Hades_(videojuego)

[14] Russ Frushtick *Hades blends God of War with Binding of Isaac in marvelous ways*. Consultado en Hades gameplay impressions: How is the new rogue-like from the creators of Bastion? - Polygon

[15] ¿Qué es la inteligencia artificial y cómo se usa? – Parlamento Europeo. Consultado en ¿Qué es la inteligencia artificial y cómo se usa? | Noticias | Parlamento Europeo (europa.eu)

[16] Ian Millington *Artificial Intelligence for Games*. San Francisco : Morgan Kaufmann, 2006, capítulo 1: Introduction, 1.1. What Is AI?

[17] Aversa, Davide *Unity Artificial Intelligence Programming: Add Powerful, Beliavable, and Fun AI Entities in Your Game with the Power of Unity.* Birmingham : Packt Publishing, Limited, 5ª edición, 2022

[18] Georgios N. Yannakakis, Julian Togelius *Artificial Intelligence and Games*. Cham: Springer International Publishing : Imprint : Springer, 2018

[19] Patricio Letelier *¿Qué tienes en tu Backlog?*. Consultado en Agility at work: ¿Qué tienes en tu Backlog (y en tus Sprints)? (agilismoatwork.blogspot.com)

[20] Patricio Letelier *Gestión de requisitos basada en pruebas de aceptación*, Material de apoyo en asignaturas PSW y PIN

[21] David Erosa García *¿Qué es Unity?*. Consultado en Qué es Unity y características principales | OpenWebinars

[22] Macondo *Unity: Qué es y cómo funciona*. Consultado en Unity: ¿Qué es y cómo funciona? – Unity

[23] Paseo por el lenguaje C# - Microsoft Learn. Consultado en Un paseo por C#: información general | Microsoft Learn

[24]Yúbal Fernándex *Qué es Github y qué es lo que le ofrece a los desarrolladores*. Consultado en Qué es Github y qué es lo que le ofrece a los desarrolladores (xataka.com)

[25] Comenzar con GitHub Desktop – Documentación de GitHub. Consultado en Comenzar con GitHub Desktop - Documentación de GitHub

[26] Lucidspark – Mapas conceptuales a Full. Consultado en ▷ Lucidspark para Mapas Conceptuales: Qué es, Ventajas, Costos (mapasconceptualesafull.com)

[27] ¿Qué es Jira Software? – Jira. Consultado en Bienvenido a Jira Software | Atlassian

[28] Patrón Observer – IONOS Digital Guide. Consultado en https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-el-patron-observer/

[29] Javier Sepúlveda Verdugo *Usar Delegates y Events en Unity*. Consultado en Usar Delegates y Events en Unity - Adictos al trabajo

[30] Patrón Singleton – IONOS Digital Guide. Consultado en https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-singleton/

Apéndices

ANEXO

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.

Al haberse desarrollado un videojuego de acción Roguelike para este TFG no creo que tenga una gran relación con los ODS ni un efecto sumamente significativo para su cumplimiento. Aún así, si que se pueden recalcar algunas relaciones entre Black Friday War y los ODS.

En primer lugar, habría que destacar el lore o argumento de Black Friday War, que bajo un corte satírico nos presenta una sociedad futura sumida en la decadencia y el caos por culpa de las continuas crisis internacionales entre los distintos países. Esto puede servir de advertencia de lo que podría suceder si nuestras naciones continúan distanciándose y enfrascándose en conflictos inútiles, en vez de unirse y enfrentarse conjuntamente a los desafíos del futuro, por lo que podría relacionarse en cierta media con el ODS 17 referente a las Alianzas.

Por otra parte, también podemos encontrar una fuerte relación con el ODS 10, de reducción de desigualdades, y el ODS 2, de hambre cero. En el futuro representado en Black Friday War, la subida de los precios provoca que la gente apenas pueda comprar lo necesario para sobrevivir, teniendo pocas oportunidades para obtener el sustento. Esto puede verse como aviso de lo que podría suceder si los precios no dejan de subir, aumentando las desigualdades sociales, lo que también puede provocar que los menos afortunados tengan más dificultades a la hora de conseguir el sustento necesario para sobrevivir.

También podemos relacionar el videojuego con el ODS 16, referente a las sociedades pacíficas. En Black Friday War, tanto los protagonistas como los enemigos son compradores que tratan de sobrevivir, estando dispuestos a todo (como demuestran los combates del juego). Esto también puede verse como una advertencia de lo violenta que puede llegar a convertirse nuestra sociedad si no tenemos cuidado, haciendo hincapié en la necesidad de construir una sociedad más pacífica e inclusiva.

Por último, podría mencionarse el ODS 5, referente a la igualdad de género. En Black Friday War existe una presencia equilibrada de personajes masculinos y femeninos, tanto entre los enemigos como dentro del grupo protagonista que controla el jugador.

Funcionamiento de la librería MonoBehaviourTree

MonoBehaviourTree es una librería open source disponible en la Unity Asset Store que ofrece. Dicha librería ofrece una solución para la implementación de la Inteligencia Artificial mediante el uso de los Behaviour Trees.

Entre las características que ofrece dicha librería, debemos destacar las siguientes:

- Integración con Unity: al ser una librería disponible en la Asset Store de Unity, resulta fácil integrarla en un proyecto de Unity sin ningún tipo de problema.
- Editor visual: MonoBehaviorTree cuenta con un editor visual que permite diseñar los BT arrastrando y uniendo los distintos nodos.
- Componente Blackboard: La Blackboard es un componente que funciona como una especie de memoria guardando el valor de una serie de variables que utiliza una IA. Aunque esta no es la primera librería que ofrece este tipo de componente (BehaviorBricks también cuenta con una Blackboard), el uso de la Blackboard de MonoBehaviourTree resulta más sencillo e intuitivo, teniendo además la posibilidad de conectar una Blackboard con otra Blackboard de forma jerárquica (siendo una el padre y la otra la hija), lo que puede ser útil a la hora de crear una memoria común para todas las IA.
- Selección aleatoria de comportamiento: Los nodos Sequence y Selector de MonoBehaviourTree tienen una opción que permite la selección aleatoria del nodo hijo a ejecutar. También encontramos un nodo decorador que decide de forma aleatoria ejecutar o no al nodo hijo.
- Una librería extensible: MonoBehaviourTree permite la creación de nuevos nodos. También permite la creación de nuevos tipos de variables y referencias para la Blackboard.
- Reusabilidad: Además de poder usar un nodo en más de una misma BT, también se posibilita el uso de Subtrees (Subárboles), es decir, BTs conectadas a otras BTs.
- Posibilidad de visualizar la ejecución de una BT: Al ejecutar el proyecto, MonoBehaviourTree te permite ver cómo se ejecuta el BT en el editor visual, pudiendo colocar Breakpoints en los nodos, paralizando la ejecución del proyecto al llegar a dicho nodo (lo que permite estudiar la ejecución de más detenidamente).
- Una buena documentación, que se puede encontrar en la página de GitHub de la librería. Además, cuenta con un activo foro en la Unity Asset Store.

Para hablar del funcionamiento de esta librería, en primer lugar, debemos hablar sobre el componente MonoBehaviourTree, que es el que contiene el BT. Cada vez que hay crear un nuevo BT, hay que añadirse dicho componente al GameObject que queremos que tenga el BT.

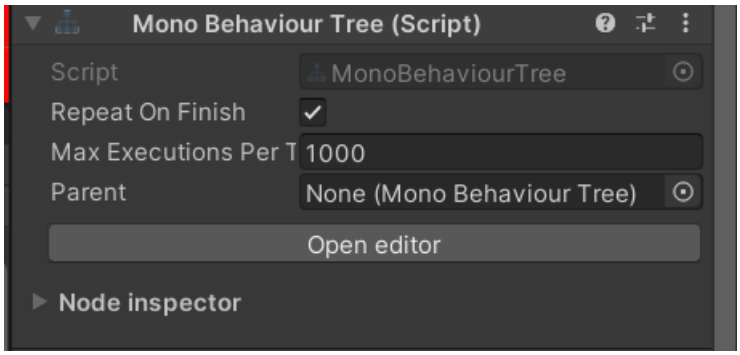


Figura 1: Vista del componente MonoBehaviourTree

Tras añadir el MonoBehaviourTree al GameObject, ya es posible editar el BT, haciendo click para ello en el botón Open editor que aparece en el componente MonoBehaviourTree en el inspector de Unity.

Figura 2: Vista del editor visual de BTs

Tras ello, ya se puede editar el BT moviendo, uniendo o eliminando nodos. Para añadir nuevos nodos, simplemente se debe hacer clic derecho sobre el editor abriendo un menú en el que

se muestran todos los nodos disponibles.

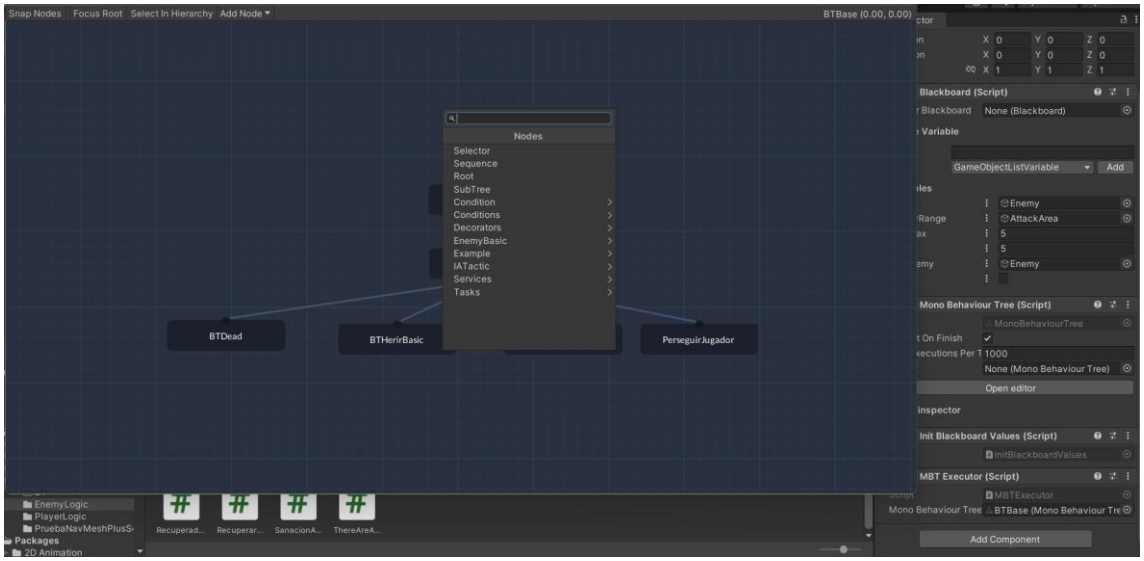


Figura 3: Editor visual con la lista de selección de nodos

Para configurar un determinado nodo, hay que seleccionarlo en el editor y entonces se podrá ver y modificar la configuración de dicho nodo desde el inspector de nodos que contiene el componente MonoBehaviourTree.

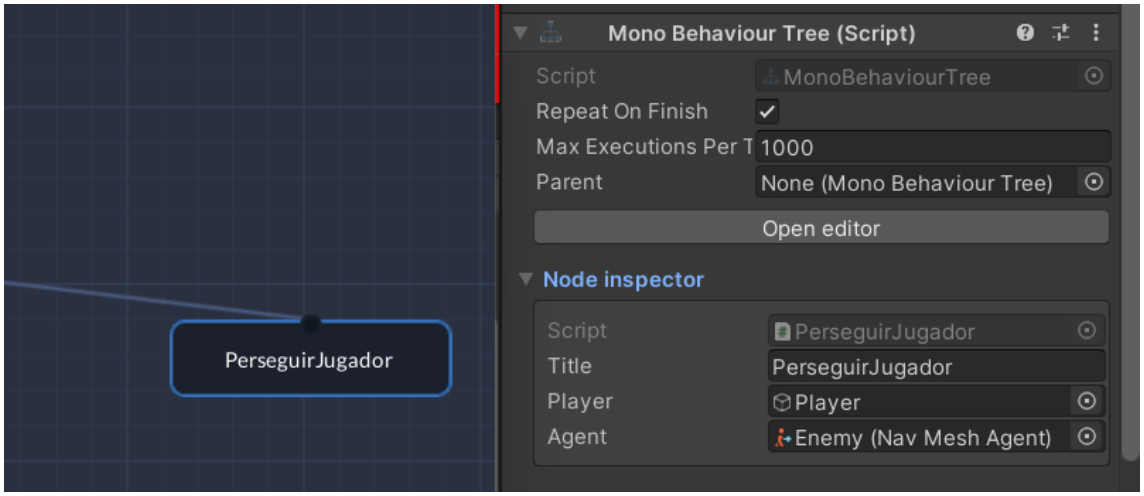


Figura 4: Vista del inspector de nodos

En cuanto a los tipos de nodos que podemos encontrar en MonoBehaviourTrees, encontramos los conocidos nodos Compuestos (Sequence y Selector), los nodos Hoja (Leaf) que representan las acciones, los nodos Decorador (Decorator) y los nodos Condición (Condition).

Hay que destacar que, en esta solución, los nodos Condition no son nodos Hoja, recordando bastante a los nodos Decorator, ya que los nodos Condition de MonoBehaviourTree deben tener un nodo hijo. Dicho nodo hijo se ejecutará dependiendo de si se cumple o no la condición que evalúa el nodo Condition. Obsérvese el siguiente ejemplo, en el que se puede observar un BT con un nodo Condition, que dependiendo del valor de la variable vida ejecutará o no el nodo DeadEnemy.

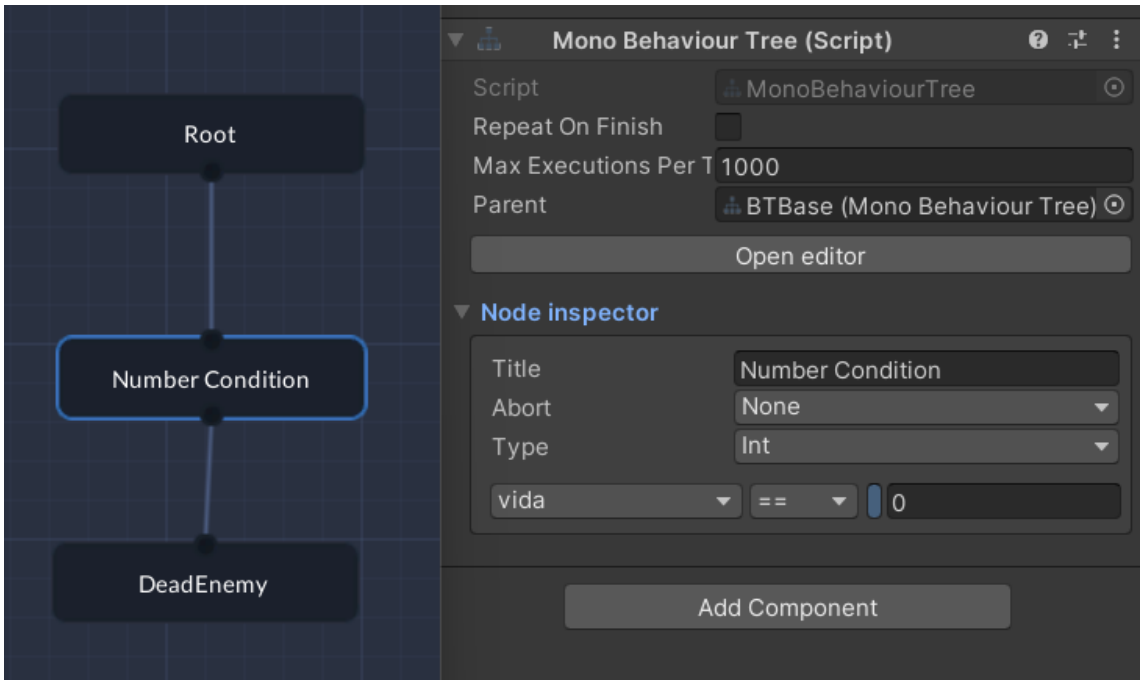


Figura 5: Ejemplo de uso de un nodo Condition

Además de estos nodos, también podemos encontrar una serie de nodos propios de la librería que merecen una atención especial:

- Nodo Root. Es el equivalente al nodo Start que se vio en los ejemplos presentados en el apartado del Estado del arte. Dicho nodo es obligatorio, teniendo que haber siempre únicamente uno en un BT, siendo padre de todos los nodos del BT.
- Nodo Service. Este tipo de nodo podría definirse como un nodo Decorator que no modifica la salida del nodo hijo. Un Service lo que hace es ejecutar una determinada acción mientras el nodo hijo se ejecuta.
- Nodo Subtree. El nodo Subtree es un nodo que contiene una referencia a otro BT. Este nodo permite conectar la BT que contiene el nodo al que hace referencia, permitiendo que el flujo de ejecución vaya desde la primera BT (que sería la BT padre) a la segunda BT (que sería la BT hijo). Es decir, el nodo Subtree es el que permite el uso de subárboles o árboles anidados, lo que facilita la reutilización de bloques de nodos.

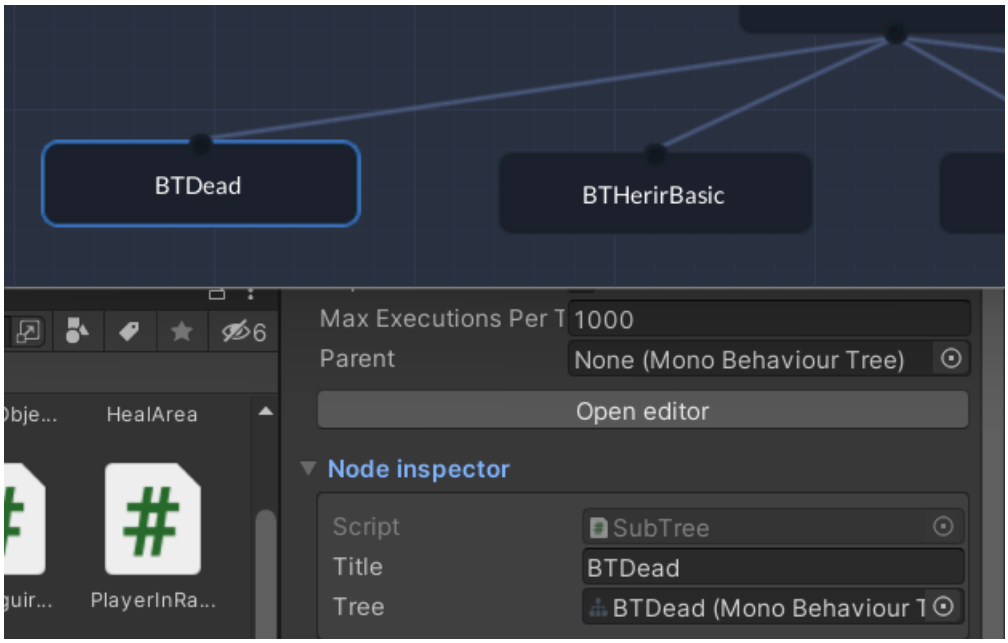


Figura 6: Ejemplo de uso de nodo Subtree

Ahora que ya conocemos los tipos de nodos disponibles, debemos explicar cómo se crean nuevos nodos. Para ello basta con crear una clase que herede de la clase base del tipo de nodo que se desea crear (Leaf, Condition, Decorator o Service) y seguir la API de la librería para su correcta implementación. Sirva de ejemplo, el siguiente fragmento de código con la implementación de un nodo Leaf.

```
[AddComponentMenu("")]
[MBTNode(name = "EnemyBasic/DamageEnemy")]
Script de Unity (7 referencias de recurso) | 0 referencias
public class DamageEnemy : Leaf
{
    public IntReference vida;
    public BoolReference herido;
    public BoolReference recuperadoDeAtaque;
    2 referencias
    public override NodeResult Execute()
    {
        if (vida == null || herido == null) return NodeResult.failure;
        vida.Value--;
        herido.Value = false;
        if (recuperadoDeAtaque != null)
        {
            recuperadoDeAtaque.Value = true;
        }
        return NodeResult.success;
    }
}
```

Figura 7: Ejemplo de código de un nodo Leaf

Observemos las dos primeras líneas. Con la línea AddComponentMenu(“”) se evita que el script con el código del nodo pueda añadirse como componente a un GameObject. En cuanto a MBTNode(name=”…”), dicha línea registra el nodo en el editor visual con un nombre, pudiendo posteriormente añadirlo a cualquier BT. En este caso, el nodo se registra con el nombre DamageEnemy incluyendo el nodo en la categoría EnemyBasic.

Por último, debemos hablar de dos componentes más de la librería: el componente MBTExecutor y el componente Blackboard. El componente MBTExecutor simplemente ejecuta el BT de un MonoBehaviourTree, mientras que la Blackboard, como ya se mencionó anteriormente, funciona como una memoria que guarda una serie de variables.

Se puede acceder a la Blackboard de MonoBehaviourTree por los distintos nodos de la BT, aunque para ello la Blackboard debe encontrarse en el GameObject en el que está el MonoBehaviourTree o en el padre del GameObject que contiene el MonoBehaviourTree.

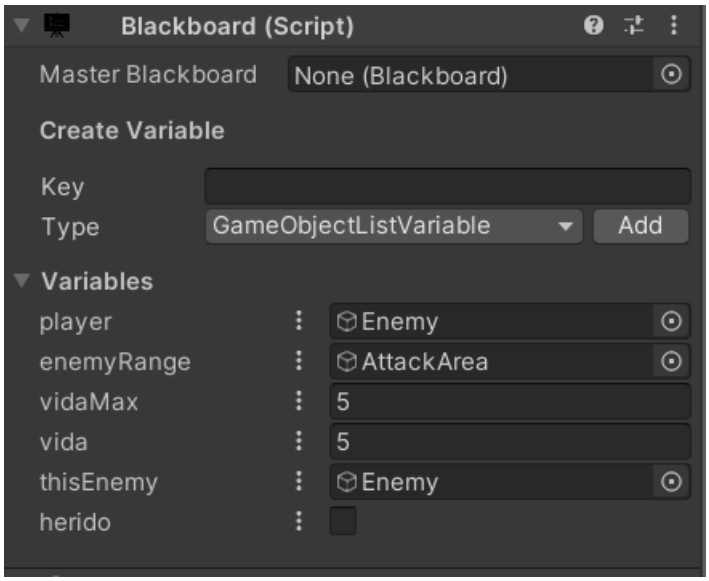


Figura 8: Vista del componente Blackboard

La Blackboard se basa en el uso de la clase Variable y la clase VariableReference. La clase Variable funciona como contenedor de datos y la clase VariableReference sirve para acceder a dicha Variable. Observemos la Leaf que hemos visto anteriormente, que cuenta con tres atributos de tipo VariableReference, más concretamente de tipo IntReference y BoolReference, lo que permite asignarles una variable de la Blackboard a través del NodeInspector.

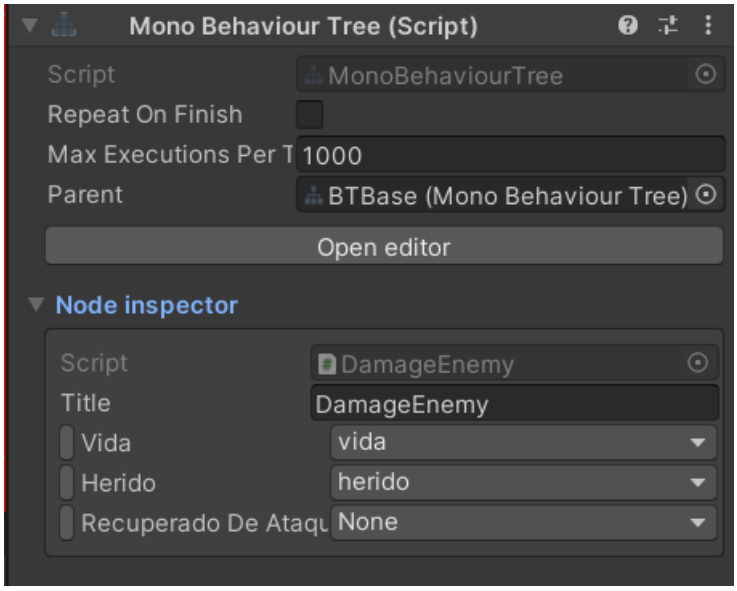


Figura 9: Ejemplo de asignación de variables a un nodo

MonoBehaviourTree, ofrece además la posibilidad de crear nuestros propios tipos de variables para la Blackboard extendiendo las clases Variable y VariableReference. Sirva de ejemplo el siguiente fragmento de código en el que se crea una Variable para poder guardar en la Blackboard una lista de GameObjects.

```
[AddComponentMenu("")]
Script de Unity (6 referencias de recurso) | 7 referencias
public class GameObjectListVariable : Variable<List<GameObject>>
{
    2 referencias
    protected override bool ValueEquals(List<GameObject> val1, List<GameObject> val2)
    {
        return val1.Equals(val2);
    }
}

[System.Serializable]
15 referencias
public class GameObjectListReference : VariableReference<GameObjectListVariable, List<GameObject>>
{
    // You can create additional constructors and Value getter/setter
    // See FloatVariable.cs as example

    // If your variable is reference type you might want constant validation
    // protected override bool isConstantValid
    // {
    //     get { return constantValue != null; }
    // }
}
```

Figura 10: Ejemplo de creación de Variable propia para la Blackboard

NavMeshPlus

NavMeshPlus es un paquete disponible en GitHub que extiende las herramientas de NavMesh de Unity, ofreciendo una serie de componentes que permiten la generación de NavMesh para escenas 2D.

Para poder generar el NavMesh 2D, se ha de añadir a la escena un GameObject con el componente NavigationSurface. También será necesario añadir el componente NavMeshCollectSources2d en el que habrá que hacer clic sobre la opción Rotate Surface to XY para poder generar correctamente el NavMesh.

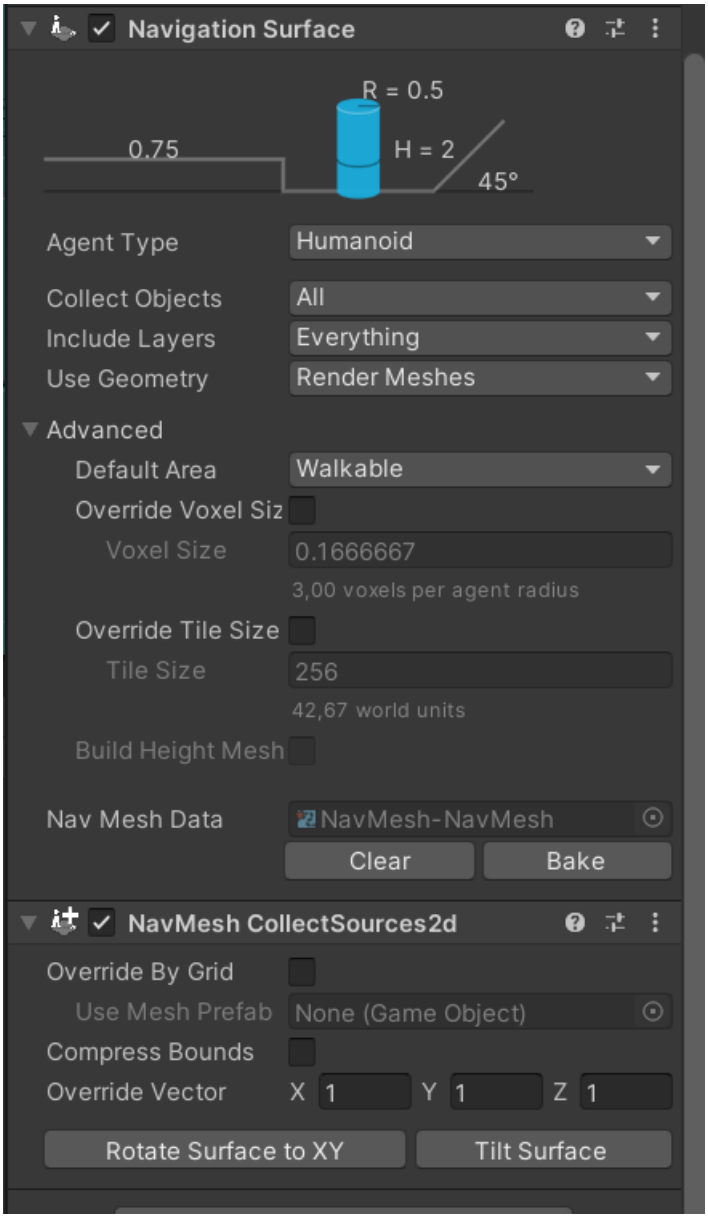


Figura 11: Vista de los componentes NavigationSurface y NavMeshCollectSources2d

Tras ello, simplemente hay que añadir a los GameObject que deseamos incluir en la generación del NavMesh el componente NavigationModifier marcando si queremos que el GameObject sea caminable (Walkable), no caminable (Not Walkable) o que se pueda saltar (Jump). En nuestro caso, dicho componente se ha añadido a los distintos Tilemaps que componen el nivel del juego, marcando como Walkable los Tilemaps del suelo y los objetos del suelo, y como Not Walkable los Tilemaps de las paredes y objetos del escenario.

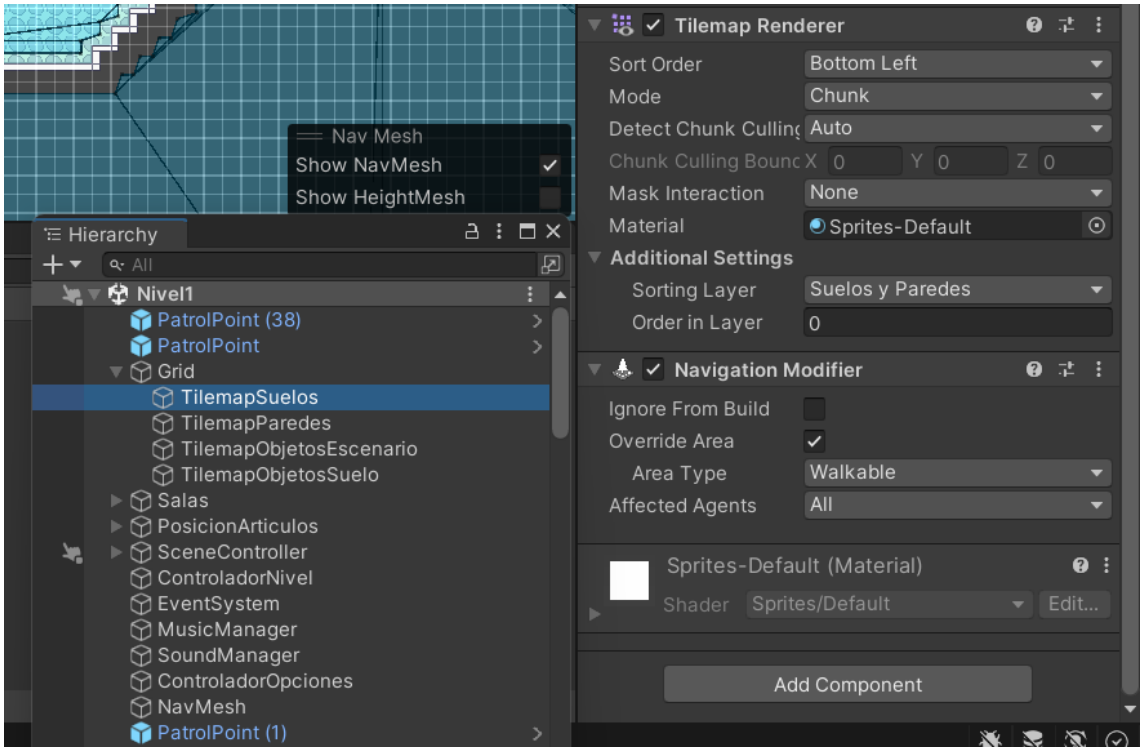


Figura 12: Componente NavigationModifier en TilemapSuelos

Tras terminar de configurar los Navigation Modifier, lo único que falta es hacer click en la opción Bake del NavigationSurface y, si todo se ha hecho correctamente, el NavMesh de la escena se generará. Obsérvese la siguiente imagen en la que se puede observar el NavMesh del nivel de Black Friday War.

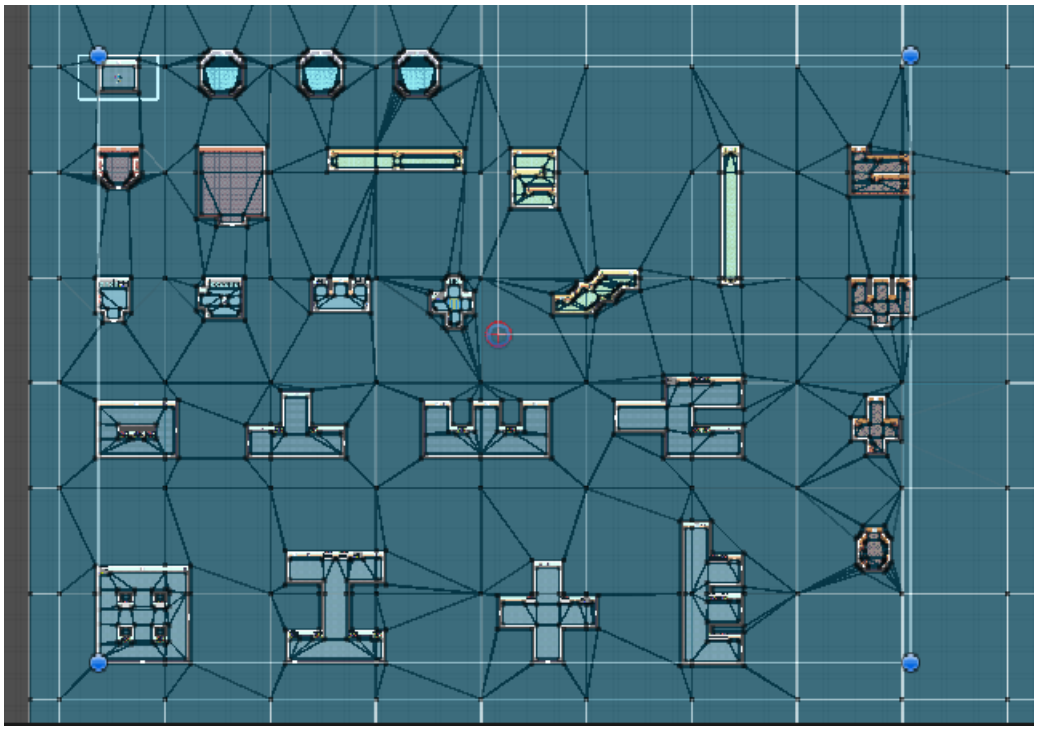


Figura 13: NavMesh del nivel

De esta forma, al colocar el componente NavMeshAgent a los enemigos, estos ya podrán desplazarse por el nivel sin ningún problema. Eso sí, para que el componente NavMeshAgent funcione correctamente en un NavMesh 2D hay que modificar algunas opciones a través de código, tal y como puede verse en el siguiente fragmento.

```
protected void initNavAgent()
{
    _agent = GetComponent<NavMeshAgent>();
    _agent.updateRotation = false;
    _agent.updateUpAxis = false;
}
```

Figura 14: Inicialización de NavMeshAgent

Pruebas de Aceptación de las Unidades de Trabajo

Pruebas de Aceptación	
Código	**Nombre**
MeleeSandbox1	Persecución del jugador
MeleeSandbox2	Dañar jugador con ataque melee
MeleeSandbox3	Asustar Enemigo
MeleeSandbox4	Buscar apoyo
MeleeSandbox5	Curación con Sanador
MeleeSandbox6	Animación con Animador
MeleeSandbox7	“Desasutar” con el tiempo
MeleeSandbox8	“Desasustar” al ser dañado durante Animación
MeleeSandbox9	“Desasustar” al ser dañado durante Curación
MeleeSandbox10	“Desasustar” cuando el Sanador es atacado durante la Curación
MeleeSandbox11	“Desasustar” cuando el Animador es atacado durante la Animación
MeleeSandbox12	Dañar Enemigo
MeleeSandbox13	Dejar Apoyo asignado si está muerto
MeleeSandbox14	Muerte del Enemigo
DistanciaSandbox1	Persecución del jugador
DistanciaSandbox2	Ataque a distancia
DistanciaSandbox3	Buscar trayectoria de lanzamiento válida
DistanciaSandbox4	Asustar Enemigo
DistanciaSandbox5	Buscar apoyo
DistanciaSandbox6	Curación con Sanador
DistanciaSandbox7	Animación con Animador
DistanciaSandbox8	“Desasutar” con el tiempo

DistanciaSandbox9	“Desasustar” al ser dañado durante Animación
DistanciaSandbox10	“Desasustar” al ser dañado durante Curación
DistanciaSandbox11	“Desasustar” cuando el Sanador es atacado durante la Curación
DistanciaSandbox12	“Desasustar” cuando el Animador es atacado durante la Animación
DistanciaSandbox13	Dañar Enemigo
DistanciaSandbox14	Dejar Apoyo asignado si está muerto
DistanciaSandbox15	Muerte del Enemigo
SanadorSandbox1	Huir cuando el jugador se acerca
SanadorSandbox2	Patrullar por la zona mientras se encuentren el jugador y otros enemigos lejos
SanadorSandbox3	Ir por el Enemigo asustado más cercano
SanadorSandbox4	Sanar Enemigo asustado
SanadorSandbox5	No sanar Enemigo no asustado
SanadorSandbox6	Interrumpir Sanación al recibir daño
SanadorSandbox7	Ignorar Enemigo asustado con Sanador asignado
SanadorSandbox8	Ignorar Enemigo asustado con Animador asignado
SanadorSandbox9	Regresar a comportamiento normal si el Enemigo asignado muere
SanadorSandbox10	Dañar Enemigo
SanadorSandbox11	Muerte del Enemigo
AnimadorSandbox1	Huir cuando el jugador se acerca
AnimadorSandbox2	Patrullar por la zona mientras se encuentren el jugador y otros enemigos lejos
AnimadorSandbox3	Ir por el Enemigo asustado más cercano
AnimadorSandbox4	Animar Enemigo asustado
AnimadorSandbox5	No animar Enemigo no asustado

AnimadorSandbox6	Interrumpir Animación al recibir daño
AnimadorSandbox7	Ignorar Enemigo asustado con Sanador asignado
AnimadorSandbox8	Ignorar Enemigo asustado con Animador asignado
AnimadorSandbox9	Regresar a comportamiento normal si el Enemigo asignado muere
AnimadorSandbox10	Dañar Enemigo
AnimadorSandbox11	Muerte del Enemigo
LímiteAnimado1	Salida del estado animado
EmbestidaSandbox1	Persecución del jugador
EmbestidaSandbox2	Iniciar embestida
EmbestidaSandbox3	Embestir jugador
EmbestidaSandbox4	Embestida fallida
EmbestidaSandbox5	Retirada post-embestida
EmbestidaSandbox6	Aturdimiento
EmbestidaSandbox7	Invulnerabilidad durante embestida
EmbestidaSandbox8	No iniciar embestida si hay obstáculos cerca del Enemigo
EmbestidaSandbox9	No iniciar embestida si no hay una trayectoria limpia hacia el jugador
EmbestidaSandbox10	Asustar Enemigo
EmbestidaSandbox11	Buscar apoyo
EmbestidaSandbox12	Curación con Sanador
EmbestidaSandbox13	Animación con Animador
EmbestidaSandbox14	Salida del estado Animado
EmbestidaSandbox15	“Desasutar” con el tiempo
EmbestidaSandbox16	“Desasustar” al ser dañado durante Animación
EmbestidaSandbox17	“Desasustar” al ser dañado durante Curación

EmbestidaSandbox18	“Desasustar” cuando el Sanador es atacado durante la Curación
EmbestidaSandbox19	“Desasustar” cuando el Animador es atacado durante la Animación
EmbestidaSandbox20	Dañar Enemigo
EmbestidaSandbox21	Dejar Apoyo asignado si está muerto
EmbestidaSandbox22	Muerte del Enemigo
AbuelaSandbox1	Muerte de Abuela Boss
AbuelaSandbox2	Dañar Abuela Boss
AbuelaSandbox3	Persecución del jugador
AbuelaSandbox4	Ataque a distancia normal
AbuelaSandbox5	Ataque a melee normal
AbuelaSandbox6	Entrada en el estado furia
AbuelaSandbox7	Ataque a distancia furia
AbuelaSandbox8	Ataque a melee furia
NerdSandbox1	Dañar Nerd Boss
NerdSandbox2	Muerte del Nerd
NerdSandbox3	Ataque de área (Descartado)
NerdSandbox4	Invocar aliados
NerdSandbox5	Nerd invulnerable mientras haya aliados vivos
NerdSandbox6	Entrada en el estado furia
NerdSandbox7	Nerd genera drones especiales
NerdSandbox8	Lanzar dron al pasar por punto intermedio
NerdSandbox9	Movimiento de los drones especiales al tocar obstáculos
NerdSandbox10	Dron especial ataca cuerpo a cuerpo al jugador
NerdSandbox11	Dron especial lanza proyectiles hacia el jugador

NerdSandbox12	Movimiento del Nerd
NerdSandbox13	Ataque especial de los drones
NerdSandbox14	Lanzar drones durante la carga de movimiento
NerdSandbox15	Dron normal desaparece al chocar con obstáculo
NerdSandbox16	Dron normal ataca al jugador
NerdSandbox17	Destruir dron normal al recibir ataque del jugador
NerdSandbox18	Dañar dron especial
NerdSandbox19	Destruir dron especial
MadreHijoSandbox1	Dañar Madre
MadreHijoSandbox2	Dañar Hijo
MadreHijoSandbox3	Hijo solicita protección al recibir daño con frecuencia
MadreHijoSandbox4	Preparar entrada en el estado protección
MadreHijoSandbox5	Movimiento en pareja durante la protección
MadreHijoSandbox6	Madre recibe daño del hijo durante la protección
MadreHijoSandbox7	Madre ataca durante la protección (Descartado)
MadreHijoSandbox8	Hijo ataca durante la protección
MadreHijoSandbox9	Finalizar protección al pasar el tiempo
MadreHijoSandbox10	Finalizar protección si Madre muere
MadreHijoSandbox11	Lanzamiento normal de la Madre
MadreHijoSandbox12	Lanzamiento normal del Hijo
MadreHijoSandbox13	Preparación del lanzamiento especial
MadreHijoSandbox14	Lanzamiento especial con hijo
MadreHijoSandbox15	Muerte de la Madre
MadreHijoSandbox16	Muerte del Hijo

MadreHijoSandbox17	Lanzamiento comba normal (Descartado)
MadreHijoSandbox18	Sujeción comba normal (Descartado)
MadreHijoSandbox19	Lanzamiento comba normal falla (Descartado)
MadreHijoSandbox20	Ataque rotatorio del hijo normal (Descartado)
MadreHijoSandbox21	Preparación del ataque especial comba (Descartado)
MadreHijoSandbox22	Ataque especial comba Madre e Hijo (Descartado)
MadreHijoSandbox23	Entrada de la Madre en el estado furia
MadreHijoSandbox24	Lanzamiento especial Madre furia
MadreHijoSandbox25	Entrada del Hijo en el estado furia
MadreHijoSandbox26	Lanzamiento continuo del Hijo en estado furia
MadreHijoSandbox27	Sujeción comba furia (Descartado)
MadreHijoSandbox28	Lanzamiento comba falla en estado furia (Descartado)
MadreHijoSandbox29	Ataque especial comba estado furia (Descartado)
MadreHijoSandbox30	Rebote de los proyectiles
MadreHijoSandbox31	Desaparición de los proyectiles con el tiempo
MadreHijoSandbox32	Los proyectiles dañan al jugador
DistanciaFinal1	Generación del Enemigo a Distancia
DistanciaFinal2	Dañado por ataque a Melee de la familia
DistanciaFinal3	Dañado por ataque de la Hija
DistanciaFinal4	Dañado por ataque a Distancia de la familia
DistanciaFinal5	Generación de ítems al morir
DistanciaFinal6	Atacar familia
DistanciaFinal7	Persecución de la familia

DistanciaFinal8	Cambiar animaciones del Enemigo según el movimiento
DistanciaFinal9	Se puede salir de la sala tras la muerte del Enemigo
DistanciaFinal10	Asustar Enemigo
DistanciaFinal11	Curación con Sanador
DistanciaFinal12	Animación con Animador
DistanciaFinal13	“Desasustar” con el tiempo
DistanciaFinal14	“Desasustar” al ser dañado durante animación
DistanciaFinal15	“Desasustar” al ser dañado durante curación
DistanciaFinal16	“Desasustar” cuando Sanador atacado durante la curación
DistanciaFinal17	“Desasustar” cuando Animador atacado durante la animación
DistanciaFinal18	Dañar Enemigo
DistanciaFinal19	Huir de la familia
DistanciaFinal20	Asignar Sanador
DistanciaFinal21	Asignar Animador
DistanciaFinal22	Huir al morir el Sanador asignado
DistanciaFinal23	Huir al morir el Animador asignado
DistanciaFinal24	Cambio de Sanador al morir el Sanador asignado
DistanciaFinal25	Cambio de Animador al morir el Animador asignado
DistanciaFinal26	Cambio a Animador al morir el Sanador asignado
DistanciaFinal27	Cambio a Sanador al morir el Animador asignado
DistanciaFinal28	No sale del estado asustado si tiene Apoyo asignado
DistanciaFinal29	Muerte del Enemigo

DistanciaFinal30	Mejorar Enemigo durante estado animado
DistanciaFinal31	Enemigo animado no puede asustarse
DistanciaFinal32	Enemigo sale del estado animado
DistanciaFinal33	Retroalimentación visual del proceso de animación (Descartado, se dejó para la UT de animaciones y efectos)
DistanciaFinal34	Retroalimentación visual del proceso de curación
DistanciaFinal35	Retroalimentación visual del estado animado (Descartado, se dejó para la UT de animaciones y efectos)
DistanciaFinal36	Retroalimentación visual del estado asustado (Descartado, se dejó para la UT de animaciones y efectos)
DistanciaFinal37	Animación del movimiento de los proyectiles
DistanciaFinal38	Animación de desaparición de los proyectiles
DistanciaFinal39	Los proyectiles lanzados desaparecen al colisionar con obstáculos o con la familia
DistanciaFinal40	Los proyectiles lanzados no hacen daño al usar la habilidad de la madre
MeleeFinal1	Generación del Enemigo a Melee
MeleeFinal2	Dañado por ataque a Melee de la familia
MeleeFinal3	Dañado por ataque de la Hija
MeleeFinal4	Dañado por ataque a Distancia de la familia
MeleeFinal5	Generación de ítems al morir
MeleeFinal6	Atacar familia
MeleeFinal7	Persecución de la familia
MeleeFinal8	Cambiar animaciones del Enemigo según el movimiento
MeleeFinal9	Sincronizar animación de ataque con rutina de ataque (Descartado, la animación de ataque se dejó para la UT de animaciones y efectos)

MeleeFinal10	Se puede salir de la sala tras la muerte del Enemigo
MeleeFinal11	Asustar Enemigo
MeleeFinal12	Curación con Sanador
MeleeFinal13	Animación con Animador
MeleeFinal14	“Desasustar” con el tiempo
MeleeFinal15	“Desasustar” al ser dañado durante animación
MeleeFinal16	“Desasustar” al ser dañado durante curación
MeleeFinal17	“Desasustar” cuando Sanador atacado durante la curación
MeleeFinal18	“Desasustar” cuando Animador atacado durante la animación
MeleeFinal19	Dañar Enemigo
MeleeFinal20	Huir de la familia
MeleeFinal21	Asignar Sanador
MeleeFinal22	Asignar Animador
MeleeFinal23	Huir al morir el Sanador asignado
MeleeFinal24	Huir al morir el Animador asignado
MeleeFinal25	Cambio de Sanador al morir el Sanador asignado
MeleeFinal26	Cambio de Animador al morir el Animador asignado
MeleeFinal27	Cambio a Animador al morir el Sanador asignado
MeleeFinal28	Cambio a Sanador al morir el Animador asignado
MeleeFinal29	No sale del estado asustado si tiene Apoyo asignado
MeleeFinal30	Muerte del Enemigo
MeleeFinal31	Mejorar Enemigo durante estado animado

MeleeFinal32	Enemigo animado no puede asustarse
MeleeFinal33	Enemigo sale del estado animado
MeleeFinal34	Retroalimentación visual del proceso de animación (Descartado, se dejó para la UT de animaciones y efectos)
MeleeFinal35	Retroalimentación visual del proceso de curación
MeleeFinal36	Retroalimentación visual del estado animado (Descartado, se dejó para la UT de animaciones y efectos)
MeleeFinal37	Retroalimentación visual del estado asustado (Descartado, se dejó para la UT de animaciones y efectos)
MeleeFinal38	Ataque a melee no daña con la habilidad de la madre activa
SanadorFinal1	Generación del Enemigo Sanador
SanadorFinal2	Dañado por ataque a Melee de la familia
SanadorFinal3	Dañado por ataque de la Hija
SanadorFinal4	Dañado por ataque a Distancia de la familia
SanadorFinal5	Generación de ítems al morir
SanadorFinal6	Cambiar animaciones del Enemigo según el movimiento
SanadorFinal7	Se puede salir de la sala tras la muerte del Enemigo
SanadorFinal8	Dañar Enemigo
SanadorFinal9	Huir de la familia
SanadorFinal10	Muerte del Enemigo
SanadorFinal11	Huir cuando la familia se acerca
SanadorFinal12	Patrullar por la zona
SanadorFinal13	Asignarse objetivo de Sanación
SanadorFinal14	Sanar enemigo asustado
SanadorFinal15	No sanar enemigos sin asustar

SanadorFinal16	Interrumpir Sanación al recibir daño
SanadorFinal17	Ignorar Enemigo asustado si ya tiene Sanador asignado
SanadorFinal18	Ignorar Enemigo asustado si ya tiene Animador asignado
SanadorFinal19	Regresar a comportamiento normal si objetivo asignado muere
AnimadorFinal1	Generación del Enemigo Animador
AnimadorFinal2	Dañado por ataque a Melee de la familia
AnimadorFinal3	Dañado por ataque de la Hija
AnimadorFinal4	Dañado por ataque a Distancia de la familia
AnimadorFinal5	Generación de ítems al morir
AnimadorFinal6	Cambiar animaciones del Enemigo según el movimiento
AnimadorFinal7	Se puede salir de la sala tras la muerte del Enemigo
AnimadorFinal8	Dañar Enemigo
AnimadorFinal9	Huir de la familia
AnimadorFinal10	Muerte del Enemigo
AnimadorFinal11	Huir cuando la familia se acerca
AnimadorFinal12	Patrullar por la zona
AnimadorFinal13	Asignarse objetivo de Animación
AnimadorFinal14	Animar enemigo asustado
AnimadorFinal15	No animar enemigos sin asustar
AnimadorFinal16	Interrumpir Animación al recibir daño
AnimadorFinal17	Ignorar Enemigo asustado si ya tiene Sanador asignado
AnimadorFinal18	Ignorar Enemigo asustado si ya tiene Animador asignado
AnimadorFinal19	Regresar a comportamiento normal si objetivo asignado muere

EmbestidaFinal1	Generación del Enemigo Embestida
EmbestidaFinal2	Dañado por ataque a Melee de la familia
EmbestidaFinal3	Dañado por habilidad de la hija
EmbestidaFinal4	Dañado por ataque a distancia de la familia
EmbestidaFinal5	Generación de ítems al morir
EmbestidaFinal6	Persecución de la familia
EmbestidaFinal7	Cambiar animaciones del Enemigo dependiendo del movimiento (Descartado, se asignó para la UT de animaciones y efectos)
EmbestidaFinal8	Se puede salir de la sala al matar al Enemigo
EmbestidaFinal9	Asustar Enemigo
EmbestidaFinal10	Curación con Sanador
EmbestidaFinal11	Animación con Animador
EmbestidaFinal12	“Desasustar” con el tiempo
EmbestidaFinal13	“Desasustar” al ser dañado durante animación
EmbestidaFinal14	“Desasustar” al ser dañado durante curación
EmbestidaFinal15	“Desasustar” cuando el Sanador es atacado durante sanación
EmbestidaFinal16	“Desasustar” cuando el Animador es atacado durante animación
EmbestidaFinal17	Dañar Enemigo
EmbestidaFinal18	Huir de la familia
EmbestidaFinal19	Asignar Sanador
EmbestidaFinal20	Asignar Animador
EmbestidaFinal21	Huir al morir el Sanador asignado
EmbestidaFinal22	Huir al morir el Animador asignado
EmbestidaFinal23	Cambio de Animador al morir el Animador asignado

EmbestidaFinal24	Cambio a Animador al morir el Sanador asignado
EmbestidaFinal25	Cambio de Sanador al morir el Sanador asignado
EmbestidaFinal26	Cambio a Sanador al morir el Animador asignado
EmbestidaFinal27	No sale del estado asustado si tiene apoyo asignado
EmbestidaFinal28	Muerte del Enemigo
EmbestidaFinal29	Mejorar Enemigo durante estado Animado
EmbestidaFinal30	Enemigo animado no puede asustarse
EmbestidaFinal31	Enemigo sale del estado animado
EmbestidaFinal32	Retroalimentación visual del proceso de curación
EmbestidaFinal33	La embestida no daña con la habilidad de la madre activa
EmbestidaFinal34	Iniciar embestida
EmbestidaFinal35	Embestir familia
EmbestidaFinal36	Embestida fallida
EmbestidaFinal37	Retirada post-embestida
EmbestidaFinal38	Aturdimiento
EmbestidaFinal39	Invulnerabilidad durante embestida
EmbestidaFinal40	No iniciar embestida si obstáculos cerca del Enemigo
EmbestidaFinal41	No iniciar embestida si no hay trayectoria limpia
EmbestidaFinal42	Durante la embestida el enemigo atraviesa otros enemigos (Descartado)
EmbestidaFinal43	Finalizar retirada con el tiempo
EmbestidaFinal44	Finalizar retirada tras ser atacado
EmbestidaFinal45	Finalizar aturdimiento con el tiempo

EmbestidaFinal46	Finalizar aturdimiento tras ser atacado
EmbestidaFinal47	Embestida golpea a un Enemigo
EmbestidaFinal48	Embestida golpea a un Enemigo Embestida que ha iniciado su Embestida
EmbestidaFinal49	Embestida golpea a un Enemigo Embestida cargando una Embestida (Descartado)
EmbestidaFinal50	Dos embestidas se cancelan al colisionar entre ellas (Descartado)
EmbestidaFinal51	No iniciar embestida si enemigos cerca del Enemigo Embestida
EmbestidaFinal52	No se asusta al recibir daño durante la retirada
EmbestidaFinal53	Puede asustarse al recibir daño durante el aturdimiento
AbuelaFinal1	Muerte de Abuela Boss
AbuelaFinal2	Dañar Abuela Boss con ataque a distancia
AbuelaFinal3	Persecución de la familia
AbuelaFinal4	Ataque a distancia normal
AbuelaFinal5	Ataque a melee normal
AbuelaFinal6	Entrada en el estado furia
AbuelaFinal7	Ataque a distancia furia
AbuelaFinal8	Ataque a Melee furia
AbuelaFinal9	Potenciación estado furia
AbuelaFinal10	Los proyectiles lanzados desaparecen al colisionar con obstáculos o con la familia
AbuelaFinal11	Animación de movimiento de los proyectiles
AbuelaFinal12	Animación de desaparición de los proyectiles
AbuelaFinal13	Los proyectiles no dañan al jugador con la habilidad de la madre
AbuelaFinal14	Empujón de la Abuela no daña con la habilidad de la madre

AbuelaFinal15	Animación de ataque a melee (Descartado)
AbuelaFinal16	Dañar a Abuela Boss mediante ataque a melee de la familia
AbuelaFinal17	Dañar con habilidad de la hija
AbuelaFinal18	Dañar Abuela Boss
NerdFinal1	Muerte del Nerd
NerdFinal2	Invocar aliados
NerdFinal3	Nerd invulnerable mientras haya aliados vivos
NerdFinal4	Nerd entra en el estado furia
NerdFinal5	Nerd genera drones especiales
NerdFinal6	Lanzar dron al pasar por punto intermedio de ruta
NerdFinal7	Movimiento de los drones especiales al tocar obstáculos
NerdFinal8	Dron especial lanza proyectiles hacia el jugador
NerdFinal9	Dron especial ataca cuerpo a cuerpo al jugador
NerdFinal10	Movimiento del Nerd
NerdFinal11	Ataque especial de los drones
NerdFinal12	Lanzar drones durante la carga de movimiento
NerdFinal13	Dron normal desaparece al chocar con obstáculo
NerdFinal14	Dron normal ataca al jugador
NerdFinal15	Destruir dron normal con ataque a Melee del jugador
NerdFinal16	Dañar dron especial
NerdFinal17	Destruir dron especial
NerdFinal18	Destruir dron normal con ataque a distancia del jugador

NerdFinal19	Destruir dron normal con habilidad de la Hija
NerdFinal20	Dañar dron especial con ataque a Melee
NerdFinal21	Dañar dron especial con ataque a Distancia
NerdFinal22	Dañar dron especial con habilidad de la Hija
NerdFinal23	Dañar Nerd con ataque a Melee
NerdFinal24	Dañar Nerd con ataque a Distancia
NerdFinal25	Dañar Nerd con habilidad de la Hija
NerdFinal26	Drones normales no afectan a la familia cuando la habilidad de la Madre se encuentra activa
NerdFinal27	Proyectiles de los drones especiales no afectan a la familia con la habilidad de la Madre activa
NerdFinal28	El ataque cuerpo a cuerpo de los drones especiales no afecta a la familia con la habilidad de la Madre activa
NerdFinal29	Generación del Nerd
MadreHijoFinal1	Dañar Madre
MadreHijoFinal2	Dañar Hijo
MadreHijoFinal3	Hijo pide protección al recibir daño con frecuencia
MadreHijoFinal4	Preparar entrada en el estado protección
MadreHijoFinal5	Movimiento en pareja durante la protección
MadreHijoFinal6	Madre recibe daño del hijo durante la protección
MadreHijoFinal7	Hijo ataca durante la protección
MadreHijoFinal8	Finalizar protección si Madre muere
MadreHijoFinal9	Finalizar protección al pasar el tiempo
MadreHijoFinal10	Lanzamiento normal de la Madre
MadreHijoFinal11	Lanzamiento normal del Hijo
MadreHijoFinal12	Preparación del lanzamiento especial normal

MadreHijoFinal13	Lanzamiento especial normal con Hijo
MadreHijoFinal14	Muerte de la Madre
MadreHijoFinal15	Muerte del Hijo
MadreHijoFinal16	Entrada de la Madre en el estado furia
MadreHijoFinal17	Lanzamiento especial Madre furia
MadreHijoFinal18	Entrada del Hijo en el estado furia
MadreHijoFinal19	Rebote de los proyectiles
MadreHijoFinal20	Desaparición de los proyectiles con el tiempo
MadreHijoFinal21	Los proyectiles dañan al jugador
MadreHijoFinal22	Madre se mueve entre determinados puntos en su estado normal
MadreHijoFinal23	Madre persigue al jugador en el estado furia
MadreHijoFinal24	Madre dañada por ataque a distancia
MadreHijoFinal25	Madre dañada por ataque a Melee
MadreHijoFinal26	Madre dañada por habilidad de la Hija
MadreHijoFinal27	Hijo dañado por ataque a distancia
MadreHijoFinal28	Hijo dañado por ataque a Melee
MadreHijoFinal29	Hijo dañado por habilidad de la Hija
MadreHijoFinal30	Las pelotas lanzadas no afectan al jugador con la habilidad de la Madre (jugador) activa
MadreHijoFinal31	Muerte de la Madre e Hijo
MadreHijoFinal32	Generación de la Madre e Hijo
CambioNivel1	Paso del nivel 1 al 2
CambioNivel2	Paso del nivel 2 al 3
CambioNivel3	Fin del juego al terminar el nivel 3
AnimEfect1	Animación de movimiento del Nerd
AnimEfect2	Animación de movimiento del Enemigo Embestida
AnimEfect3	Animación de movimiento de la Madre (jefe)

AnimEfect4	Animación de movimiento de los drones
AnimEfect5	Animación de movimiento de las pelotas
AnimEfect6	Animación de movimiento del Hijo (jefe)
AnimEfect7	Animación de desaparición de las pelotas al golpear al jugador
AnimEfect8	Animación de desaparición de los drones normales al golpear al jugador
AnimEfect9	Animación de desaparición de los drones normales al golpear un obstáculo
AnimEfect10	Animación de desaparición de los drones especiales al quedarse sin vida
AnimEfect11	Efecto asustado en los enemigos Normales
AnimEfect12	Efecto animado en los enemigos Normales
AnimEfect13	Efecto de Animación
AnimEfect14	Sonido de lanzamiento de Drones
AnimEfect15	Sonido de lanzamiento de pelotas
AnimEfect16	Sonido de lanzamiento de proyectiles de la Abuela
AnimEfect17	Animación ataque a Melee
AnimEfect18	Sonido de los drones al desaparecer
AnimEfect19	Sonido de las pelotas al golpear al jugador
AnimEfect20	Efecto aturdido del Enemigo Embestida
Retaguardia1	El Sanador se dirige hacia la retaguardia de los atacantes respecto al jugador tras su generación
Retaguardia2	El Sanador se mueve en la retaguardia
Retaguardia3	El Sanador abandona la retaguardia al detectar Enemigo asustado
Retaguardia4	El Sanador regresa a la retaguardia tras terminar la Sanación

Retaguardia5	El Sanador se mueve cuando la retaguardia se actualiza por el movimiento del jugador y los atacantes
Retaguardia6	El Sanador se mueve cuando la retaguardia se actualiza por la muerte de un Enemigo atacante
Retaguardia7	Si no hay Enemigos atacantes, los Sanadores huyen
Retaguardia8	El Animador se dirige hacia la retaguardia de los atacantes respecto al jugador tras su generación
Retaguardia9	El Animador se mueve en la retaguardia
Retaguardia10	El Animador abandona la retaguardia al detectar Enemigo asustado
Retaguardia11	El Animador regresa a la retaguardia tras terminar la Animación
Retaguardia12	El Animador se mueve cuando la retaguardia se actualiza por el movimiento del jugador y los atacantes
Retaguardia13	El Animador se mueve cuando la retaguardia se actualiza por la muerte de un Enemigo atacante
Retaguardia14	Si no hay Enemigos atacantes, los Animadores huyen

Refactorizaciones, cambios y restructuración sobre el proyecto original

Gran parte del código del proyecto original se ha reutilizado en el nuevo proyecto, aunque han tenido que sufrir algunas modificaciones. A continuación, veremos las más relevantes.

Ataque a distancia

En la versión original del Enemigo Distancia, para el ataque a distancia se utilizaba un componente denominado DisparoEnemigo. Dicho componente administraba completamente el lanzamiento de proyectil, incluyendo el tiempo de espera entre cada lanzamiento y su lanzamiento (autogestionaba el lanzamiento). El componente principal del Enemigo Distancia simplemente accedía al componente para habilitar el ataque cuando el jugador estuviera en el rango de ataque y deshabilitar el ataque en el caso contrario.

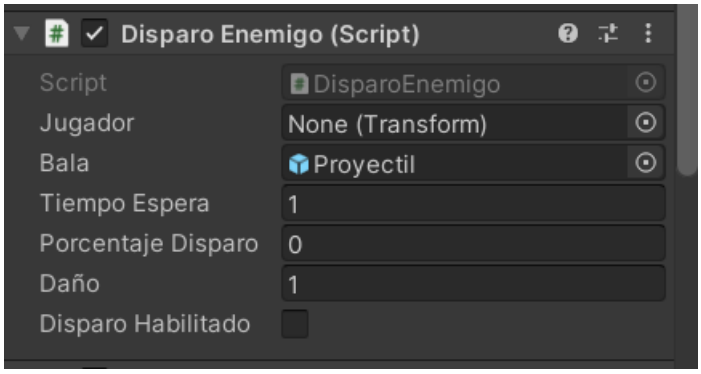


Figura 15: Componente DisparoEnemigo

Los principales cambios son que el nuevo componente utilizado para el ataque a distancia (DisparoProyectil) no lleva el control del tiempo de espera entre cada lanzamiento (dicha administración la lleva el controlador del enemigo) y el lanzamiento no es autogestionado, siendo necesario que la BT del enemigo acceda al componente DisparoProyectil para el disparo.

Control de las animaciones

En lo referente al control de las animaciones, en el proyecto original no hubo componentes específicos referentes al manejo de las animaciones, realizándose su administración en el componente principal de control del enemigo.

En nuestro proyecto, se decidió crear un componente separado centrado únicamente en las animaciones de movimiento (EnemyAnimationsManager y HelpEnemyAnimationsManager). De la misma forma, para las animaciones de ataque del Enemigo Melee, también se creó un componente centrado en su manejo (MeleeAttackAnimationController).

Generación de enemigos

Para gestionar correctamente la generación de enemigos se tuvo que realizar modificaciones sobre el HallController. En primer lugar, se tuvo que modificar la función SpawnearOleada, función que genera la oleada de enemigos en la sala. Dicha función puede observarse en el siguiente fragmento de código, en el que va utilizando uno a uno todos los Spawners de la sala para generar a los enemigos.

```
1 referencia
public void SpawnearOleada()
{
    if(Tipo!=TipoSala.SalaJefe) Nivel.musicManager.activarMusicaCombate();
    foreach (var spawner in _spawns)
    {
        var obj = spawner.SpawnearEnemigo();
        Debug.Log("Controlador sala, pos enemigo inicial: " + obj.transform.position.x + ";" + obj.transform.position.y);
        var enemigo = obj?.GetComponent<Enemigo>();
        if (enemigo != null)
        {
            enemigo.MiSala = this;
            _enemigos.Add(enemigo);
        }
    }
    ComprobarEnemigos();
}
```

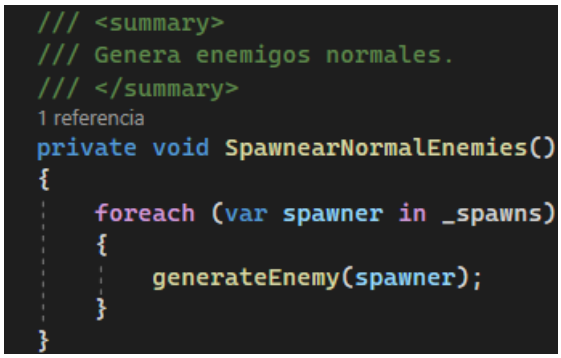
Figura 16: Función SpawnearOleada original

La modificación que se hizo fue distinguir los casos en los que se genera un jefe (es el controlador de la sala del jefe) y en los que se generan enemigos normales (es el controlador de una sala normal). En el siguiente fragmento de código puede visualizarse como quedaría la nueva función SpawnearOleada.

```
/// <summary>
/// Genera los enemigos de la sala.
/// </summary>
1 referencia
public void SpawnearOleada()
{
    if(Tipo!=TipoSala.SalaJefe) {
        Nivel.musicManager.activarMusicaCombate();
        SpawnearNormalEnemies();
    }
    else
    {
        SpawnearBoss();
    }
    ComprobarEnemigos();
}
```

Figura 17: Función SpawnearOleada modificada

Por otra parte, ambos casos siguen la misma rutina al usar un Spawner para generar un enemigo, por lo que se decidió extrapolar el código dentro del foreach de la función SpawnearOleada orignal a una función independiente que se utilice tanto para generar jefes como enemigos normales: la función generateEnemy.



```
/// <summary>
/// Genera enemigos normales.
/// </summary>
1 referencia
private void SpawnearNormalEnemies()
{
    foreach (var spawner in _spawns)
    {
        generateEnemy(spawner);
    }
}
```

Figura 18: Ejemplo de uso de la función generateEnemy en el caso de generar enemigos normales

Aplicación del patrón Observer

Ya se ha visto que el patrón Observer se ha usado en varios casos mencionados en el apartado de desarrollo. En este anexo se verán los casos en los que se ha aplicado el patrón Observer, pero no han podido estudiarse en el apartado de Desarrollo.

Comunicación de la Madre e Hijo

La Madre e Hijo necesitan comunicar cierta información entre ellos para su correcto. Por ello sus controladores se conectan mediante eventos, haciendo ambos de sujeto y observador.

En el siguiente diagrama se pueden observar los controladores de ambos enemigos y su relación mediante los eventos. Debido a que SonBossController extiende del MotherBossController comparten algunos eventos, como OnCobossDie y OnSpecialLaunchPrepared.

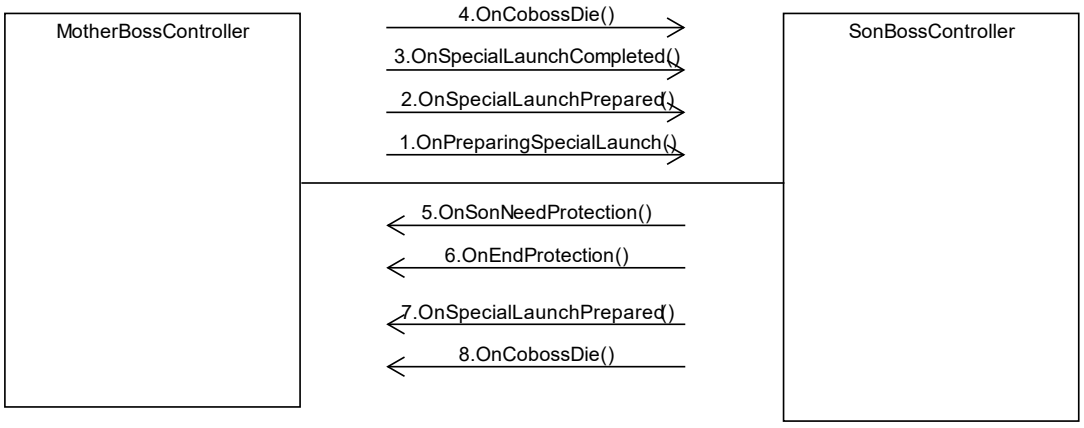


Figura 19: Comunicación entre la Madre y el Hijo

En total encontramos seis eventos diferentes: OnCobossDie (evento utilizado para comunicar la muerte de un jefe al otro jefe), OnPrepaaringSpecialLaunch (evento de la Madre para indicar que se ha iniciado la preparación del lanzamiento especial), OnSpecialLaunchPrepared (evento utilizado por ambos jefes para comunicar al otro que están listos para iniciar el lanzamiento especial), OnSpecialLaunchCompleted (evento utilizado por la Madre para indicar al hijo que ha terminado el lanzamiento especial), OnSonNeedProtection (evento que usa el Hijo para indicar a la Madre que necesita protección) y OnEndProtection (evento que usa el Hijo para indicar a la Madre que ya no necesita protección).

Destrucción de proyectiles

Para evitar que los proyectiles del Nerd y de la Madre e Hijo se queden en la sala tras la derrota del jefe y puedan dañar al jugador, se decidió suscribir los controladores de los drones y las pelotas al evento OnEnemyDie (producido al morir el enemigo) de su lanzador (el Nerd, la Madre o el Hijo) para que desaparezcan al desaparecer el jefe que las lanzó.

En este caso los observadores serían los proyectiles y el sujeto el lanzador.

En el siguiente diagrama se puede observar la conexión del controlador del dron con su lanzador, el Nerd (NerdBossController).

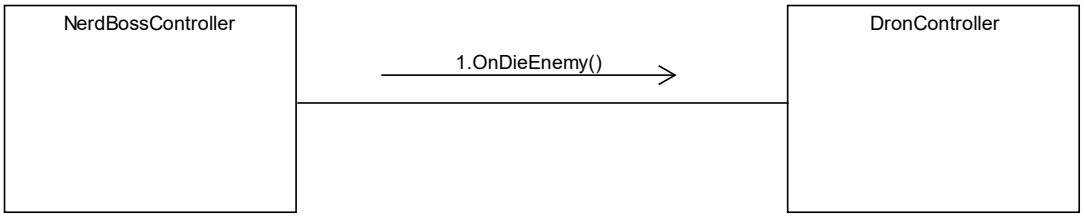


Figura 20: Comunicación entre el dron y el Nerd

En cuánto a la Madre e Hijo, en el siguiente diagrama podemos observar la conexión de la pelota (BallController) y su lanzador, que tendrá un controlador MotherBossController (recordemos que SonBossController extiende del controlador de la Madre).

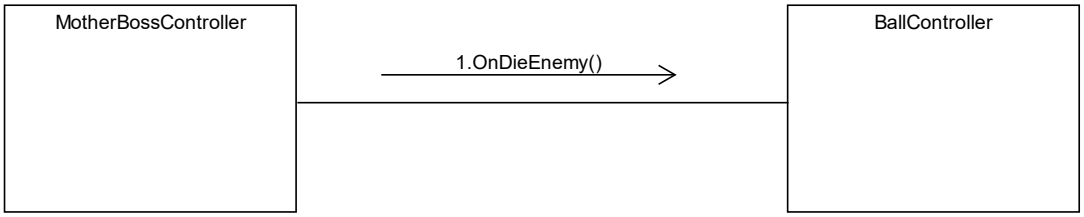


Figura 21: Comunicación entre la Madre e Hijo con la pelota

BLACK FRIDAY WAR

GAME DESIGN DOCUMENT

Fuentes García, Diego Manuel (dmfuegar@inf.upv.es)

Duart Marzo, Jorge (jduamar@inf.upv.es)

Noguera Guijarro, Francisco José (fjnoggui@inf.upv.es)

Índice:

1. Presentación

1.1. Título del juego

Black Friday War

1.2. Concepto Abstracto

Explora un peligroso centro comercial durante el Black Friday. Busca los artículos de la lista de compra mientras luchas contra hordas de consumidores enloquecidos y reúnes equipamiento para fortalecer a tu familia.

2. Planificación

2.1. Asignación de trabajo

Al tratarse de un proyecto y equipo pequeños, según vayamos avanzando iremos gestionando la asignación de las distintas tareas de forma dinámica con ayuda de alguna aplicación o herramienta externa (por ejemplo Trello o similares).

2.2. Diagrama de Gantt

15-19 Marzo	**20-26 Marzo**	**27-2 Marzo**	**3-9 Abril**	**10-16 Abril**	**17-23 Abril**	**24-30 Abril**	**1-7 Mayo**
Tomar decisiones técnicas							
Implementación de las mecánicas básicas							

		Creación y/o búsqueda del arte necesario para el juego						
			Incorporación del arte al juego					
				Implementación de características extra				
			Pruebas y correcciones					
						Preparaciones finales		

3. General

Género: Juego de acción y aventura roguelike

Plataforma: PC

Público Objetivo: Jugadores de los roguelike y/o aficionados a los juegos desafiantes

PEGI: +16 debido a la existencia de cierta violencia

Características diferenciadoras:

- La acción del juego se traslada a un lugar cercano y conocido por el jugador: un centro comercial.
- El juego contará con un humor satírico y adulto.
- La transformación de los objetos cotidianos que podemos encontrar en unos grandes almacenes en armas mortíferas.
- El jugador no controlará a un personaje solitario, algo bastante común en los roguelike, sino a un grupo de cuatro personajes, lo que también conllevará cierto toque estratégico.

4. Gameplay

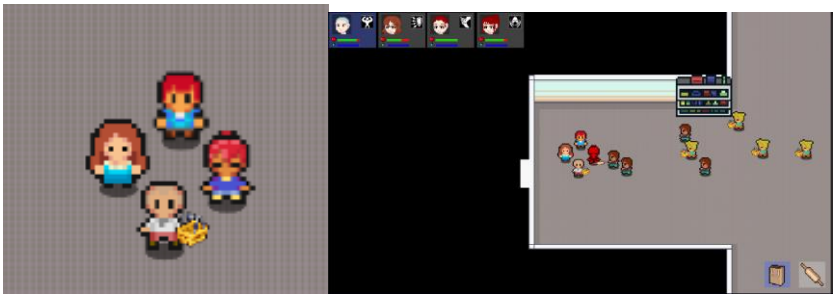
4.1. Acciones y movimiento

La familia siempre irá junta, con un miembro arriba, otro abajo, uno a la derecha y uno a la izquierda. Antes de comenzar el nivel, el jugador puede elegir como distribuir a la familia en la formación.

Una vez empezado el nivel, el jugador controlará al líder y toda su familia, pudiéndose mover libremente, apuntar y atacar. La familia seguirá al líder manteniendo la formación.

En cualquier momento puedes cambiar de líder con un botón. Hacer esto provocará que la formación rote y se ponga como líder el que se encontraba a la izquierda de este.

Depende de a qué parte de la formación ataquen, los enemigos dañarán a un miembro de la familia u a otro.



4.2. Habilidades

Cada uno de los 4 protagonistas principales, que veremos más en detalle en apartados posteriores, dispondrá de una habilidad exclusiva. Estás habilidades consumirán usos, que serán limitados y únicamente recargables con ítems, además de tener un cooldown. Las habilidades son:

- El padre será capaz de dar a la familia un aumento de fuerza que permitirá realizar daño

- La madre hará que el grupo sea invulnerable durante un pequeño periodo de tiempo

- El hijo proporcionará un aumento temporal a la velocidad del grupo

- La hija realizará un grito que dañará y alejará a los enemigos cercanos

5. Historia (lore)

5.1. Prólogo

En un futuro no muy lejano, debido al encarecimiento del nivel de vida ocasionado por las constantes crisis internacionales, las ofertas del Black Friday son la única oportunidad de adquirir los recursos necesarios para sobrevivir un año más. Todo esto ha provocado que ese día se convierta en una verdadera batalla campal por las mejores ofertas.

Los Remunson, una familia normal y corriente, deberán internarse en las profundidades del centro comercial durante el Black Friday para completar la lista de la compra y sobrevivir otro año.

5.2. Desarrollo

Los Remunson explorarán las peligrosas secciones del centro comercial buscando los artículos de su lista, enfrentándose a los enloquecidos consumidores dispuestos a matar por la mejor oferta.

5.3. Epílogo

Habrá dos finales distintos, uno bueno y otro malo, dependiendo de si el jugador gana o pierde la partida:

- El final bueno consistirá en que la familia Remunson consigue vencer todos los obstáculos y reunir los artículos de la lista de la compra, asegurándose así su supervivencia.
- El final malo consistirá en que los Remunson abandonan el centro comercial con las manos vacías tras haber sido derrotados, con sus esperanzas de sobrevivir truncadas.

6. Niveles de juego

En total, el juego contará con 3 niveles, representando cada uno una sección diferente del centro comercial, lo que se reflejará en la temática de cada nivel. Cada sección se dividirá en salas, las cuales se generarán y distribuirán de forma aleatoria. El orden de aparición de los niveles será aleatorio, en una partida el primer nivel podría ser la sección de alimentación y en otra la de electrónica, por ejemplo.

Cada sección contará con varios tipos de enemigo específicos o temáticos (es decir, enemigos que solo aparecerán en dicha sección) y un jefe, todos con un diseño acorde a la temática del nivel. También habrá armas específicas de cada sección con un diseño que refleje la temática del nivel, las cuales solo se podrán adquirir en dicho nivel.

Por otra parte, también se contará con dos tipos de enemigos genéricos (uno de ataque cuerpo a cuerpo y otro de ataque a distancia) que podrán aparecer en cualquiera de los niveles. Cada tipo contará con tres variantes, siendo la primera la más débil (aparecerá en el primer nivel) y la tercera la más fuerte (aparecerá en el último nivel). Las variantes del enemigo de ataque cuerpo a cuerpo serán el comprador matamoscas, comprador con bate de beisbol y comprador con maza, mientras que las variantes del enemigo de ataque a distancia serán el comprador con bolsa de plástico, comprador con bolsa de tela y comprador con carrito de la compra personal.

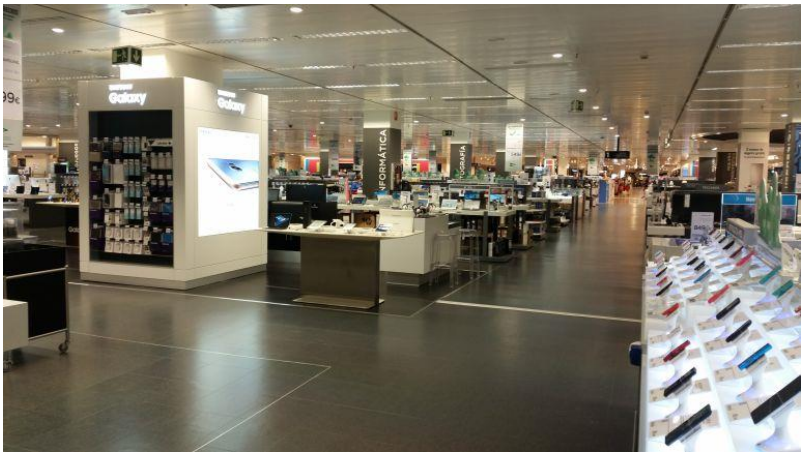
También habrá dos armas genéricas que podrán encontrarse en todos los niveles. Dichas armas consisten en el bate de beisbol (ataque cuerpo a cuerpo) y la bolsa de la compra (ataque a distancia).

Cada sección contará con tres artículos clave (acordes su temática) que el jugador deberá reunir para poder pasar a la sala del jefe y enfrentarlo. Al derrotarlo, se dará el nivel como completado.

6.1. Sección de alimentación

- Sección centrada en los productos alimenticios.
- Enemigos temáticos: el comprador con carrito, el comprador con barra de pan, el cocinero lanza tenedores y comprador con especias.
- Artículos clave que podrían aparecer en el nivel: la manzana, el brick de leche, las galletas, la pizza y la hamburguesa
- Jefe de nivel: “La Abuela”.

6.2. Sección de electrónica



- Sección centrada en los productos de electrónica.
- Enemigos temáticos: el influencer con palo selfie, el informático con ratón como látigo y el comprador con microondas.
- Artículos clave que podrían aparecer en el nivel: la televisión, el móvil, el portátil, la lavadora y la videoconsola.
- Jefe de nivel: “El Nerd”.

6.3. Sección de juguetería

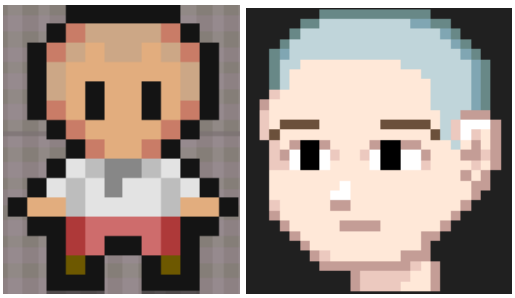
- Sección centrada en los productos de entretenimiento para niños.
- Enemigos temáticos: el niño con palo con cabeza de caballo, el niño con gran lanza bolas, el niño con globos de agua.
- Artículos clave que podrían aparecer en el nivel: el oso de peluche, los cubos con letras, el juego de mesa, la caja con bloques de construcción y el triciclo.
- Jefe de nivel: “La Madre Mimosa y El Niño Mimado”.

7. Personajes

7.1. Principales

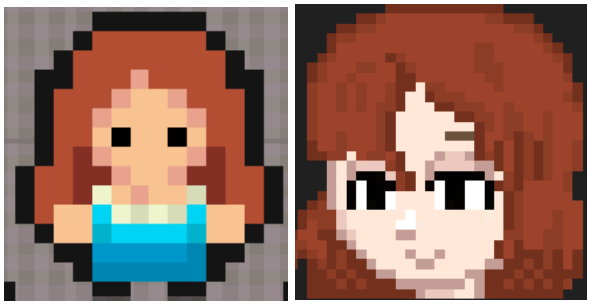
La familia Remunson

Padre



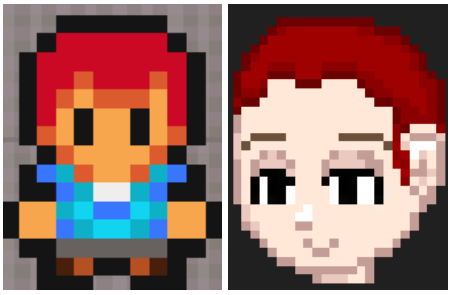
Oficinista de mediana edad que trata de mantener a su familia en tiempos difíciles. Se ha matado a trabajar el último año para ahorrar el dinero suficiente para el Black Friday. Sabe que el Black Friday es peligroso y que mucha gente ha caído en la búsqueda de la mejor oferta, pero si con ello consigue asegurar la supervivencia de su familia, está dispuesto a darlo todo, incluso su vida.

Madre



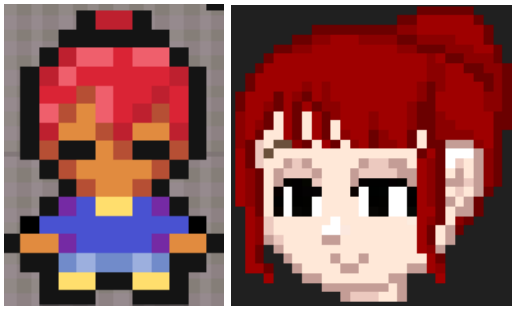
Ama de casa cuyo único objetivo es asegurar el bienestar de su familia, sobre todo de sus dos hijos. Odia la violencia y la locura que ha desatado el Black Friday, pero si es necesario, no dudará en enfrentarse al más duro de los enemigos para proteger a su familia. El deseo de proteger a sus seres queridos son la fuente de su fuerza.

Hijo



La dureza de unos tiempos inciertos también ha afectado en gran medida a los niños, y el varón de los Remunson no es una excepción. Con apenas 12 años ya es capaz de defenderse solo, incluso ha aprendido a manejar ciertas armas. Su pequeña estatura y agilidad lo convierten en un versátil combatiente en la batalla del Black Friday.

Hija



Con tan solo 8 años, la más pequeña de los Remunson ya tiene una gran comprensión del peligroso mundo que le rodea. Harta de ser protegida siempre por sus padres y su hermano mayor, ha entrenado los últimos años para demostrar su valía y ser una pieza clave en la batalla del Black Friday.

7.2. Secundarios

Conserje

Hombre mayor y misterioso que trabaja en el centro comercial como encargado de la limpieza. Siempre se mantiene neutral frente a los conflictos del Black Friday, apareciendo solo para deshacerse de los desperfectos, restos y objetos que los compradores dejan atrás al finalizar la batalla. Aparece y desaparece casi de forma mágica, dejando como única seña un cartel de “Suelo fregado”.

Compradores (Enemigos)

Todos aquellos que, al igual que los Remunson, acuden al centro comercial en busca de las mejores ofertas. Por desgracia, los productos en oferta son limitados, por lo que si desean completar su lista de la compra deberán luchar por ellos. Ellos serán el principal enemigo de la familia Remunson.

Podemos dividir a los enemigos (consumidores enloquecidos) en dos grupos principales: los que atacan cuerpo a cuerpo, y los que atacan a distancia. Además, también están aquellos más genéricos que pueden aparecer en cualquiera de las distintas secciones, y los que son específicos de una de estas.

Así, veamos los enemigos por esta última categoría:

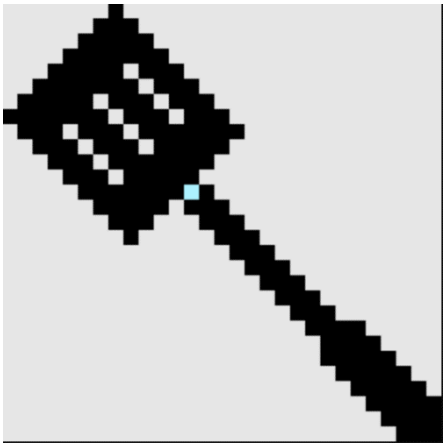
Genéricos:

Dentro de esta categoría tenemos 4 enemigos. Dos a distancia y dos cuerpo a cuerpo, de tal forma que existe una versión débil y otra fuerte para cada tipo.

Comprador con cesta de la compra:

Se trata de un comprador cualquiera que, al igual que los protagonistas, solo busca poder adquirir lo que necesita.

Comprador con matamoscas:



Algunos compradores decidieron ir preparados con algo con lo que defenderse desde un inicio, y por algún motivo todos parecían poseer algún matamoscas.

Comprador con bolsa de tela grande:

Aquellos que tuvieran una bolsa más grande que las simples de plástico no dudaron en llevársela. Gracias, a esto los artículos que podían lanzar para defenderse eran más grandes y contundentes.

Comprador con bate:

Algunos de los clientes, ante la situación de caos, decidieron defenderse con algún utensilio mejor a los simples matamoscas. En este caso, un bate parecía bastante apropiado.

Comprador con rodillo:

Los clientes más astutos vieron en el rodillo una nueva utilidad además de la de amasar. A pesar de ser un utensilio de cocina, la contundencia de los golpes de un rodillo puede dejar inconsciente a los compradores más cabeza duras, y de paso, amasarles el cráneo.

Comprador con carrito de la compra personal:

Incluso mejor que las bolsas grandes, los carritos personales permiten llevar cosas todavía más grandes que sirven como una gran defensa.

Comprador con maza:

En cuanto a armas fuertes, los clientes que en mejor forma física se encontraban eran capaces de llevar, a la máximo, una gran maza que no dudarían en usar.

Comprador sanador:

Comprador obsesionado con la salud. No puede aguantar ver como sus compañeros son heridos durante la batalla, por lo que rápidamente acude a ellos para curarlos. No ataca, pero si se deja con vida mucho tiempo, puede resultar un gran obstáculo en la batalla.

Comprador animador:

La motivación puede decaer en la batalla, y este comprador lo sabe mejor que nadie. Se ha marcado como objetivo asegurar que el ánimo de sus camaradas no baje, utilizando para ello su contagiosa actitud positiva, aumentando la velocidad y la fuerza de sus compañeros. No ataca, pero si no se elimina rápido, la batalla puede tornarse más peligrosa.

Específicos:

- Sección de alimentación:

Comprador con carrito:

Este sujeto comprobó, no solo que el carro permitía comprar más cosas a la vez, sino que este era bastante útil para abrirse paso entre el resto de los compradores. Cuando localiza a su enemigo frente a él, realiza una fuerte embestida capaz de lanzar a su enemigo contra la pared, tras lo que retrocede para evitar un contraataque. Sin embargo, durante la embestida pierde el control del carro, por lo que es incapaz de girar o retroceder en caso de que su rival le esquive, quedando aturdido y expuesto tras el fallido ataque.

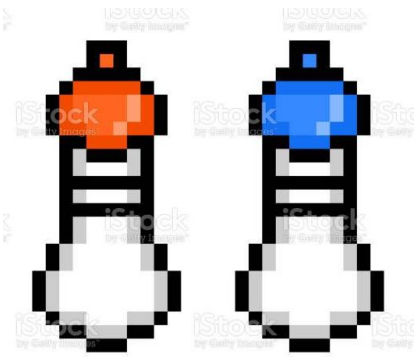
Comprador con barra de pan:

Hubo clientes que se dieron cuenta que los bates estaban funcionando bastante bien, así que decidieron improvisar algo similar con lo que tenían a mano.

Cocinero lanza tenedores:

Algunos cocineros profesionales decidieron usar algo a lo que estuvieran acostumbrados a utilizar para defenderse, y lo primero que se les ocurrió fueron los tenedores.

Comprador con especias:



El picor que produce ciertas especias, entre otros efectos, es suficiente para algunos como para usarlas en defensa propia.

- Sección de electrónica:

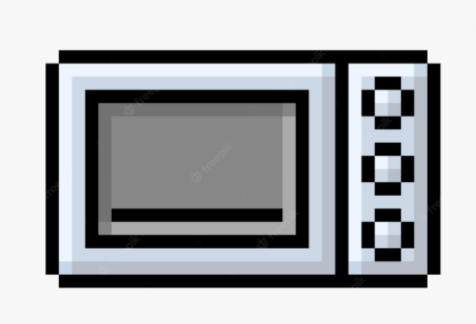
Influencer con palo de selfie:

Los influencers que andaban por la sección electrónica no podían dejar el móvil ni para pelear por su supervivencia. Eso sí, como con este no llegaban suficientemente lejos, se decantaron por usar mejor el propio palo de selfie.

Informático con ratón como látigo:

Los informáticos que buscaban nuevos componentes también necesitaban de algo para defenderse, y como los teclados estaban muy caros, prefirieron usar los ratones.

Comprador con microondas:



Todos eran conocedores de que las ondas de los microondas son muy peligrosas, así que no es de extrañar que haya compradores que intentaran aprovecharse de esta característica para sentirse más seguros.

Oficinista con relojes explosivos:

Algunos relojes eran algo inestables, y esto causaba que un golpe fuerte hiciera que estos explotarán. Los compradores que se dieron cuenta de esta propiedad la aprovecharon al máximo.

- Sección de juguetería:

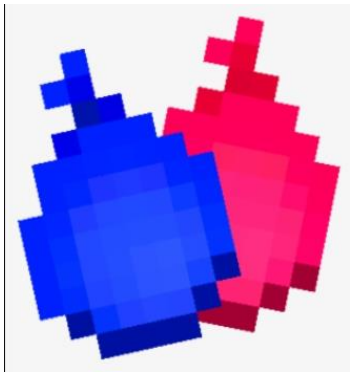
Niño con palo con cabeza de caballo:

Los niños que andaban por la juguetería también querían defenderse, y los palos con cabeza de caballo parecían cumplir bien esta función.

Niño con gran lanza bolas:

Aquellos niños que preferían no tener que acercarse para mantener a raya a los demás decidieron usar el mejor y más grande lanza bolas que está en el mercado en ese momento.

Niño con globos de agua:



Hubo niños que se decantaron por los clásicos globos de agua como arma.

Jefes de nivel

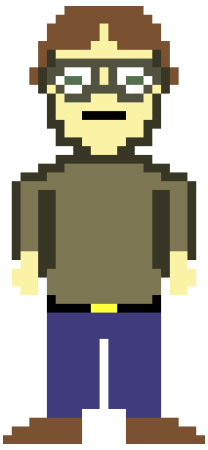
La Abuela

Jefe de la sección de alimentación. Abuela obsesionada con la buena cocina que ha reunido una gran colección de cupones para obtener prácticamente gratis todos los ingredientes de su lista. Ataca usando un rodillo y lanzando cupones.

La Abuela tiene dos patrones de comportamiento: el estado normal o inicial y el estado furia. Durante el estado normal, la Abuela tiene dos ataques, uno a distancia y otro cuerpo a cuerpo. El ataque a distancia es utilizado cuando el jugador está lejos del alcance del ataque cuerpo a cuerpo, dicho ataque consiste en lanzar 5 ráfagas de 3 proyectiles (cupones) a la familia, tras la quinta vez, debe esperar un tiempo para volver a lanzar una nueva oleada de cupones. En cuanto al ataque cuerpo a cuerpo, este consiste en golpear al grupo dando un gran empujón a la formación, alejándola de ella. El ataque cuerpo a cuerpo solo es utilizado cuando el grupo se encuentra a su alcance, y dicho ataque tarda un tiempo en hacerse efectivo (si el grupo se aleja, el ataque se cancelará).

El estado furia se activa cuando la vida de la Abuela es igual o menor a la mitad de su vida máxima. Tras ello, la velocidad de movimiento de la Abuela, la velocidad de los cupones y el daño de dichos proyectiles se duplican. Además, el ataque a distancia también se ve modificado, consistiendo ahora en 10 ráfagas de dos tipos que se alternan continuamente en la oleada, es decir, 5 ráfagas son de un tipo y las otras 5 del otro. El primer tipo es el visto en el estado normal, el lanzamiento de tres cupones, mientras que el segundo tipo consiste en el lanzamiento de 8 cupones en todas direcciones. En cuanto al ataque cuerpo a cuerpo, sigue siendo el mismo que en el estado normal, lo único que sucede es que cuando se active el ataque a melee, el ataque a distancia no se desactivará.

El Nerd



Jefe de la sección de electrónica. Fan de los dispositivos electrónicos que intenta mantenerse al día en estos tiempos tan difíciles. No dudará en enfrentar a los que puedan resultar un obstáculo para su "actualización" anual. Utiliza dispositivos electrónicos para atacar.

Los ataques del Nerd se basan en el uso de artilugios de la sección de electrónica. Uno de sus ataques se basa en el uso de un móvil para llamar a sus colegas, que acudirán a la batalla para ayudarle (se generarán enemigos aleatorios en la sala durante este ataque). También podrá lanzar ráfagas de drones que volarán en formación contra la familia. Al recibir un ataque de tipo Melee hará un pequeño ataque en área que no hará daño, pero alejará a la familia. El movimiento del Nerd podría basarse en raíles, dándole un cierto efecto de uniformidad y orden.

El movimiento del Nerd se basará en puntos de movimiento invisibles colocados en la sala a través de los que se moverá con gran velocidad (será el jefe más rápido). El Nerd tratará en todo momento de mantenerse alejado de la familia dirigiéndose a los puntos más alejados de la sala. Cada vez que pase por un eje, lanzará un dron contra la familia (que podría perseguir a la familia). Cuando llegue a su punto destino, iniciará el ataque con drones especial, tras el que se quedará en el punto un tiempo, lanzando drones según una frecuencia determinada antes de calcular una nueva ruta a otro punto destino.

Cuando el jugador esté muy cerca de él durante un tiempo, realizará un ataque en área que no dejará de hacer daño al jugador hasta que salga del área, momento en el que se desactivará el ataque.

Cada vez que pierda una determinada cantidad de puntos de vida (por ejemplo, 10 puntos) utilizará un ataque en el que invocará una cantidad aleatoria de enemigos normales (de tipo Embestida, Melee o Distancia), entre tres y cinco enemigos. Tras ello, el Nerd seguirá con sus ataques y su movimiento, pero no podrá recibir daño hasta que los enemigos invocados no sean vencidos.

Al entrar en el estado furia, cuando se reduzca su vida hasta una determinada cantidad (por ejemplo, a partir de la mitad) el Nerd generará dos drones especiales al pasar un periodo determinado de tiempo (por ejemplo, cada 20-30 segundos), que se moverán rebotando en las paredes y que lanzarán proyectiles contra la familia. Si el jugador no tiene cuidado, la sala se llenará de estos drones y la batalla se volverá un infierno.

Todos los drones pueden destruirse si reciben ataques. Los drones normales son fáciles de destruir, mientras que los especiales que rebotan con las paredes y atacan necesitan más ataques para ser destruidos.

La Madre Mimosa y El Niño Mimado

Jefe de la sección de juguetería. Dúo conformado por una madre incapaz de decir no y su pequeño hijo con el síndrome del emperador. Enfrentarán sin piedad a aquel que no les haga caso. Coordinan sus movimientos para atacar.

Los movimientos de la madre son más lentos en comparación con su hijo, cuyos movimientos son más rápidos y frenéticos. Como ataque por defecto, lanzan pelotas que pueden rebotar en las paredes, en ráfagas entre 3 y 5 pelotas. En ciertas ocasiones, la madre lanza un ataque en el que lanza ráfagas de pelotas en todas direcciones, mientras el niño se mueve a su alrededor, redirigiendo las pelotas que toque contra el jugador. Cuando el niño recibe daño con mucha frecuencia la madre acudirá en su rescate y durante un periodo de tiempo se moverán uno junto al otro, y cualquier ataque que reciban solo afectará a la madre.

El niño también puede atacar usando una especie de comba o cuerda, contando con diversos ataques: el principal, la comba gira a su alrededor haciendo daño a todo el que toque, el secundario, en el que lanza la comba contra la familia, inmovilizándola durante unos segundos, o el especial, la madre agarra el otro extremo de la comba y persiguen a la familia (durante este ataque la madre hereda la velocidad del hijo).

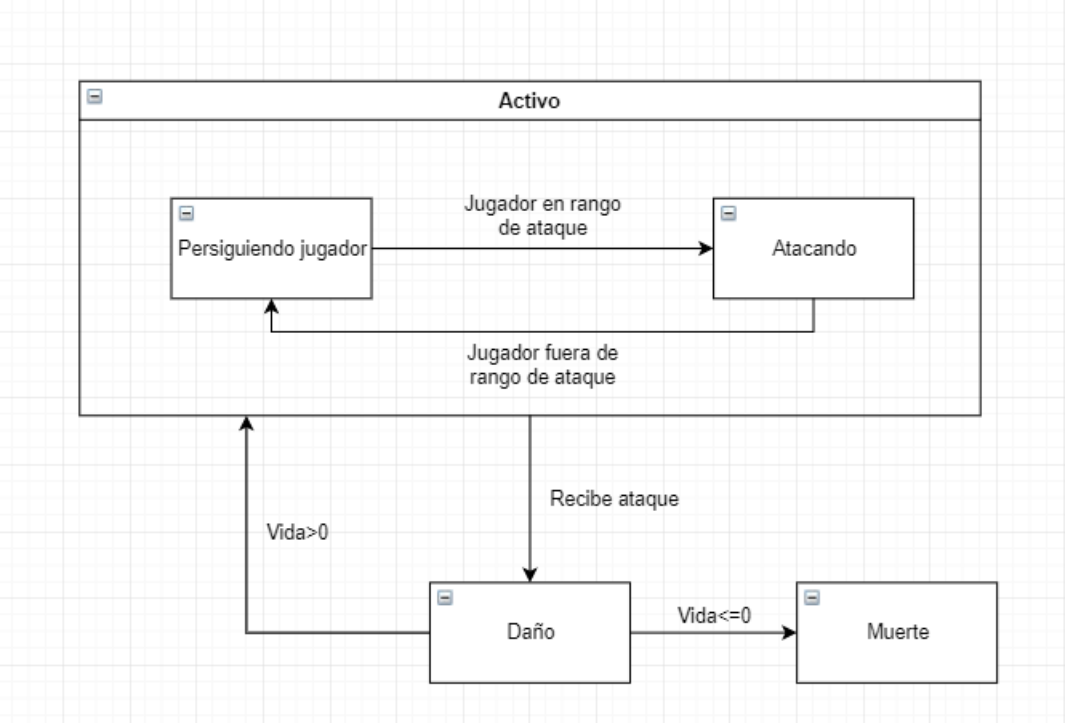
El ataque especial consta de tres embestidas con la comba, al principio de cada embestida localizan al jugador y se dirigen hacia él. Tras tocar la pared finalizan la embestida y, si aún no han llegado al límite, continúan con las embestidas.

Durante el estado de protección (cuando la madre se mueve junto al hijo), el hijo se moverá con la misma velocidad que la madre y solo podrá usar el ataque normal de la comba, mientras la madre solo será capaz de realizar su ataque a distancia normal.

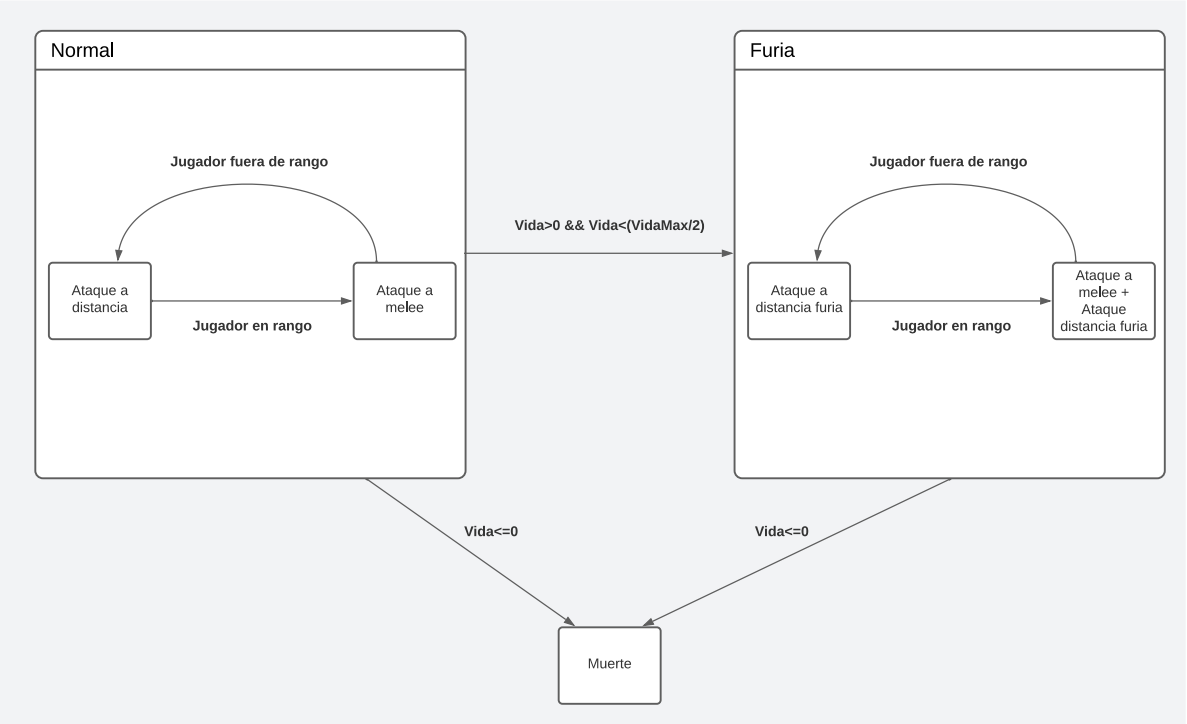
Cuando la madre es derrotada, el hijo entrará en una especie de estado furia, en el que tanto su velocidad como ataque aumentarán. Los ataques secundarios de la comba también se verán modificados: el ataque de agarre en vez de inmovilizar al jugador lo atraerá y, si es esquivado, la comba quedará enganchada a la pared y el niño tratará de tocar a la familia con la comba hasta tocar la pared. En cambio, si el niño es derrotado, será la madre la que entrará en un estado furia aumentando su fuerza y velocidad, lanzando con más frecuencia el ataque de las pelotas en todas direcciones (además, las pelotas tardarán más en desaparecer).

7.3. IA

Máquina de estados de los enemigos básicos



Máquina de estados de la Abuela (jefe de la sección de alimentación)



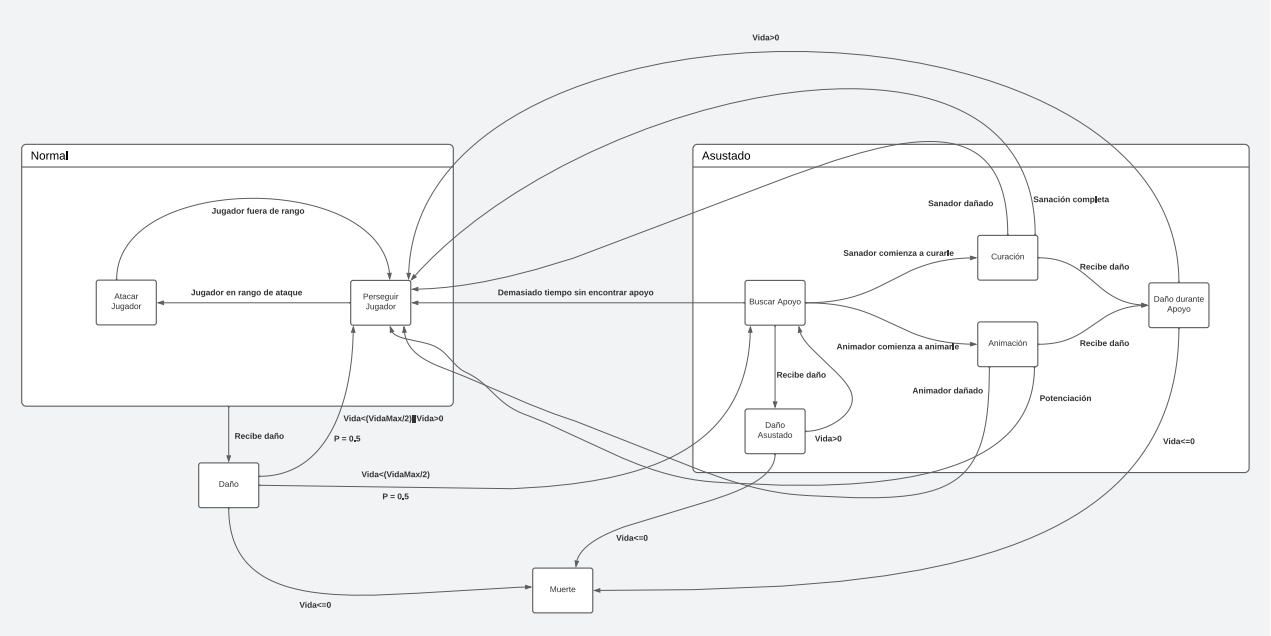
Máquina de estados enemigo a Melee actualizado

El comportamiento del enemigo con ataque a melee puede dividirse en dos patrones, el patrón Hostil y el patrón Asustado.

El enemigo comienza en el patrón Hostil, en el cual persigue el jugador hasta que el jugador se encuentra en su rango de ataque, tras lo que comienza a atacar al jugador. Cuando el jugador abandona el rango de ataque, el enemigo vuelve a perseguirlo. Cuando el enemigo recibe daño, si aún le queda vida sufre un pequeño stun tras el que, si la vida es mayor o igual a la mitad de la vida máxima, volverá a perseguir al jugador. En el caso de que la vida sea menor a la mitad de la vida máxima, habrá una probabilidad (por defecto un 50%) de que en vez de perseguir al jugador, el enemigo pase a buscar un sanador, entrando en el patrón Asustado.

En el patrón Asustado, el enemigo huye del jugador buscando un sanador. En el caso de que entre en el rango de acción de un sanador, se iniciará su recuperación, quedando inmóvil junto al sanador. La recuperación terminará cuando la vida del enemigo se haya recuperado completamente o hayan sido interrumpidos por un ataque del jugador (recibiendo daño el enemigo o el sanador). Tras salir de la recuperación, si aún le queda vida, volverá a perseguir al jugador. En el caso de que no encuentre sanador, también volverá a perseguir al jugador si un animador sube sus parámetros o lleva demasiado tiempo de búsqueda.

Si al recibir daño, la vida del enemigo se agota, se inicia su muerte.

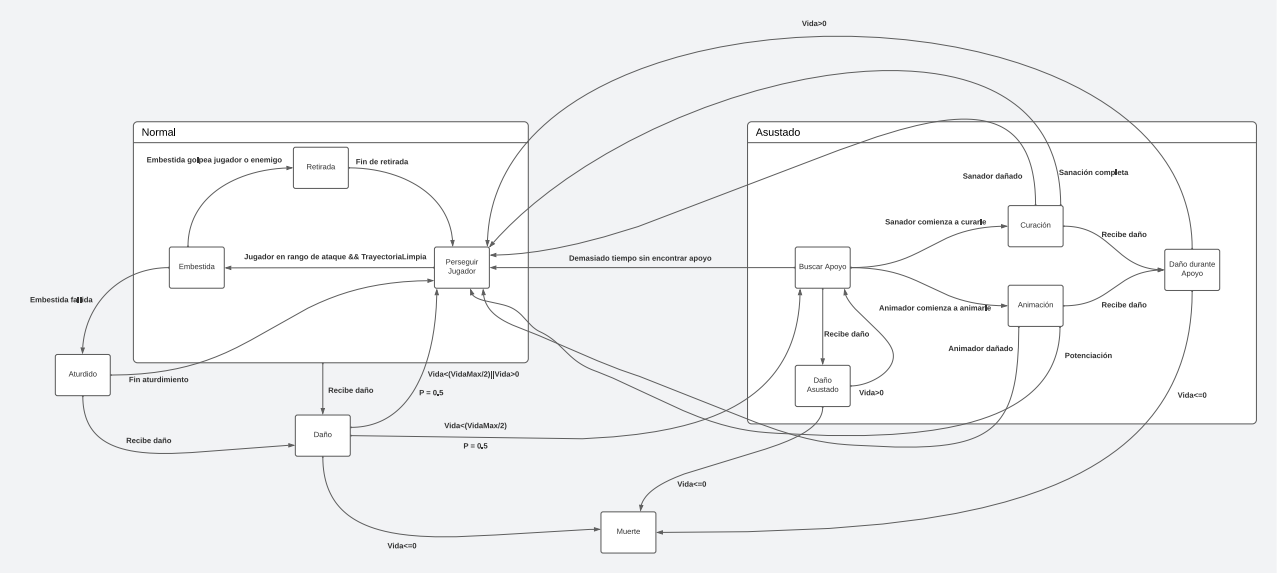


Máquina de estados enemigo con ataque de embestida (comprador con carrito)

Al igual que con el enemigo a melee, dividimos el comportamiento de este enemigo en los patrones Hostil y Asustado. El patrón Asustado es prácticamente igual al del enemigo a Melee, por lo que nos centraremos en el Hostil, que sufre algunas diferencias.

Al perseguir al jugador, cuando el jugador se encuentra dentro de una posible trayectoria para la embestida (el rango de ataque pasaría a ser una especie de línea recta), el enemigo inicia su embestida. Si la embestida da en el blanco, el enemigo inicia una retirada hasta conseguir una distancia prudencial contra el jugador, volviendo, tras ello, a perseguirlo. En el caso de que el jugador evite la embestida, tras terminar la embestida golpeando una pared o cualquier otro obstáculo, el enemigo quedará aturdido y no podrá moverse. Saldrá del aturdimiento cuando reciba daño.

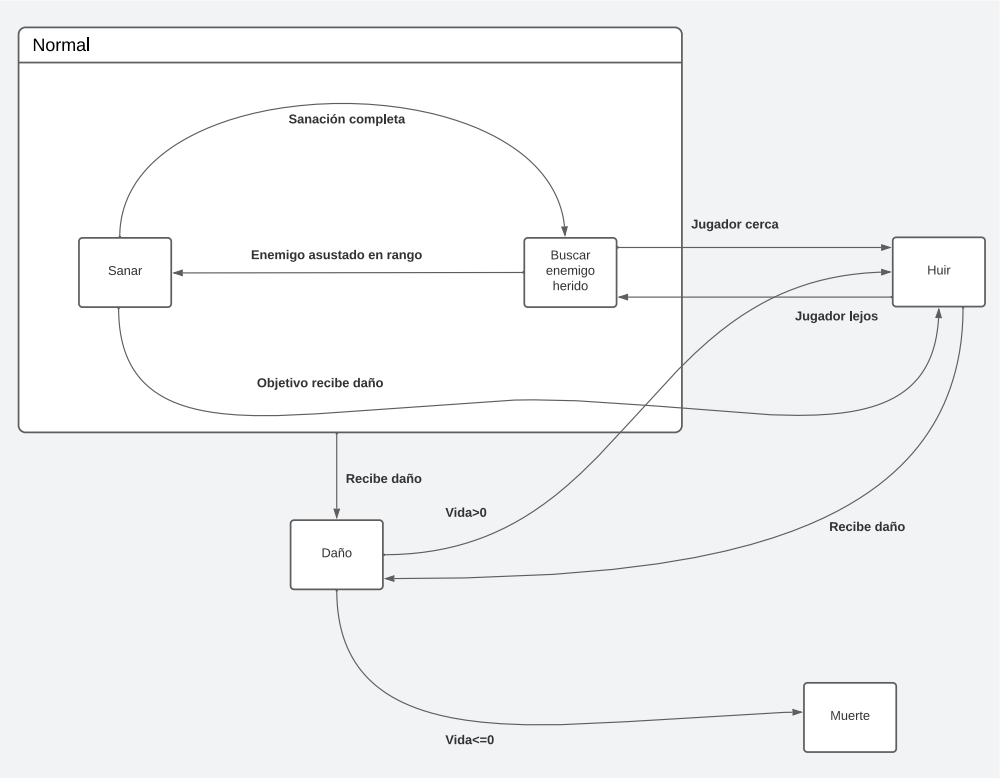
En cuanto al comportamiento del enemigo al recibir daño, sigue siendo el mismo que el del enemigo a melee.



Máquina de estados de enemigo sanador

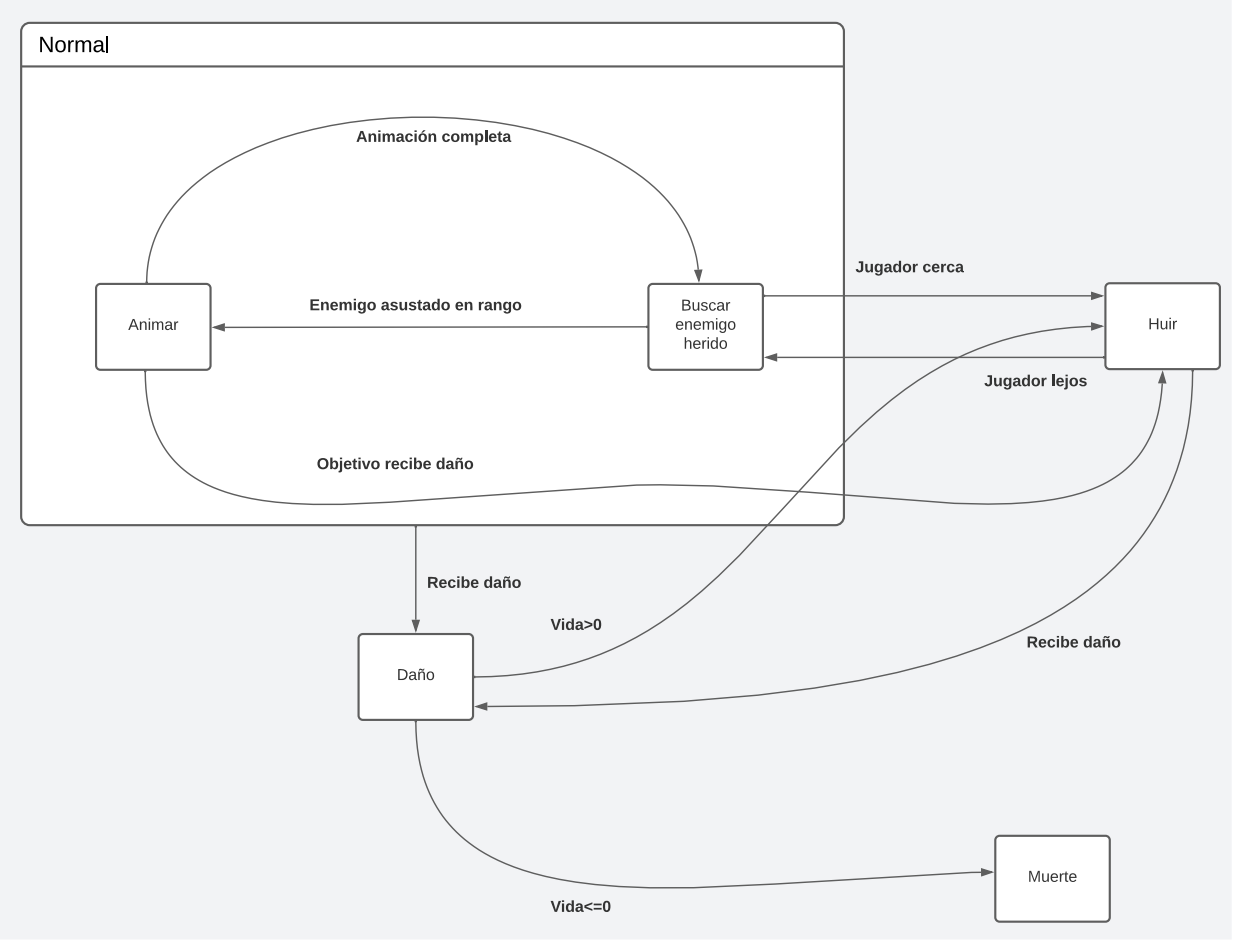
El sanador, como enemigo no hostil, tiene un comportamiento distinto a los demás. Podemos agrupar su comportamiento en el patrón Normal y Asustado.

En el patrón Normal busca compañeros que necesiten sanación. Cuando un enemigo herido entra en su rango de acción inicia el proceso de recuperación o sanación, del que solo saldrá cuando se complete la sanación o reciben un ataque. Al recibir daño, si sigue con vida, el sanador entra en el patrón Asustado, que en este caso, solo se basa en huir y alejarse del jugador. Cuando el jugador se encuentra lejos, vuelva a buscar enemigos heridos. También puede pasar al patrón Asustado si el jugador está muy cerca, aunque no reciba ningún ataque.



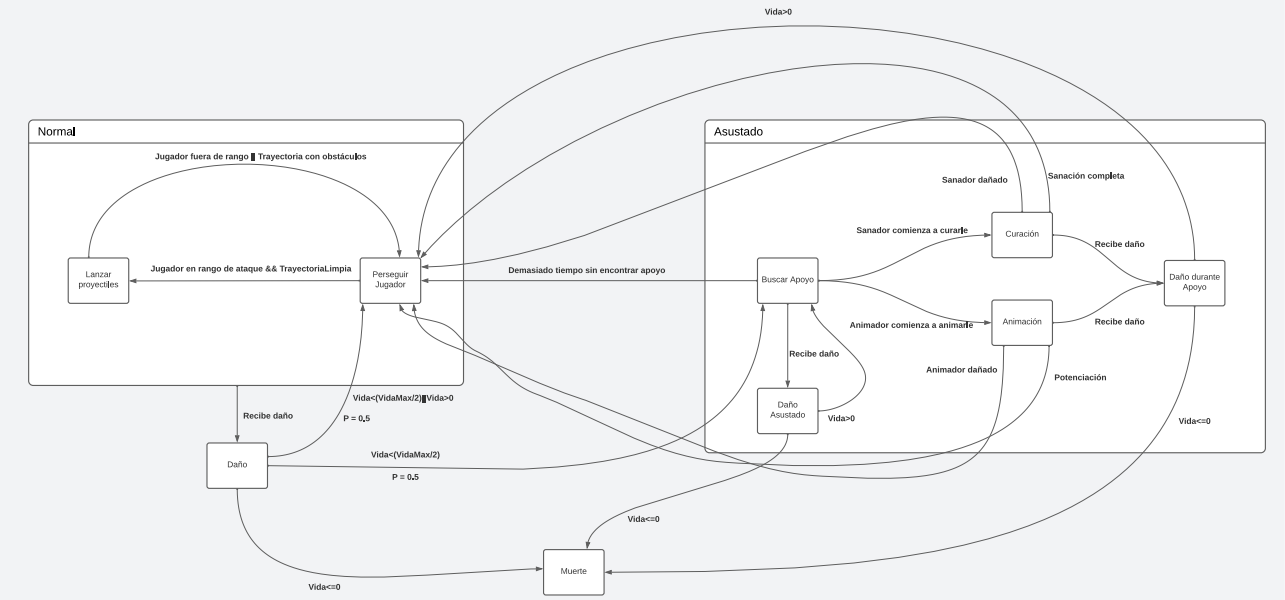
Máquina de estados de enemigo animador

El animador tiene un comportamiento similar al sanador, solo que no busca enemigos heridos, sino a cualquier enemigo, y la sanación se sustituye por la “animación” (un aumento de los estados del objetivo) siendo este proceso mucho más rápido que la sanación.

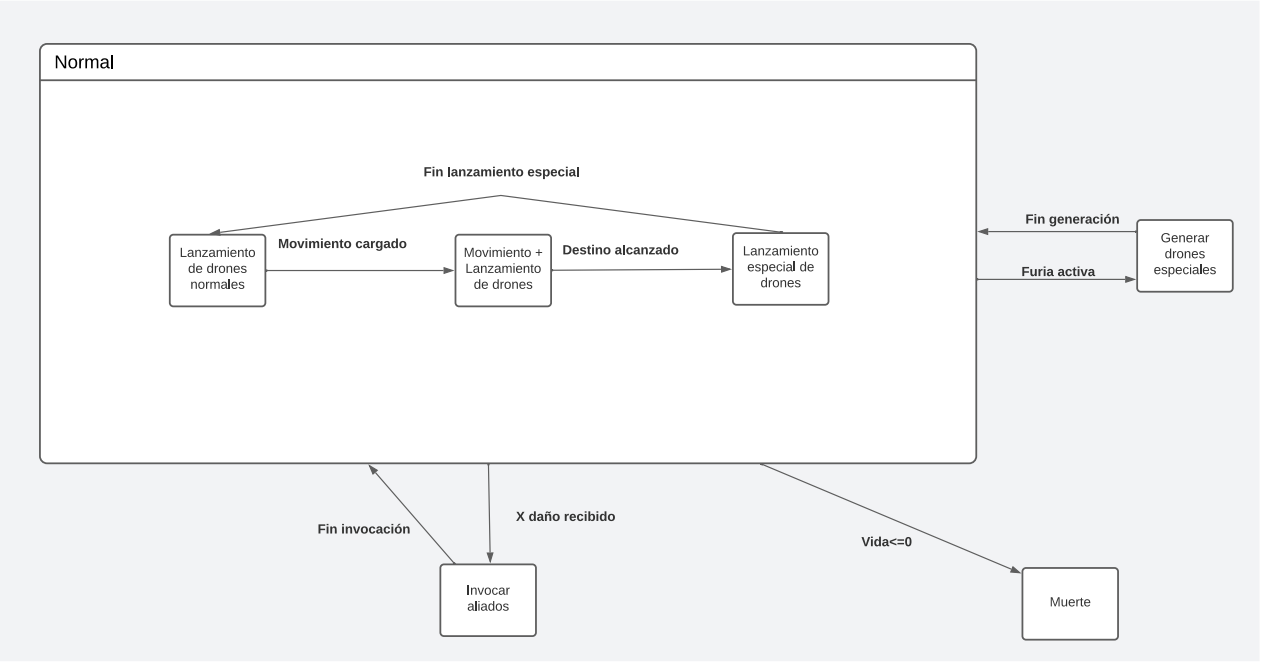


Máquina de estados enemigo a distancia actualizado

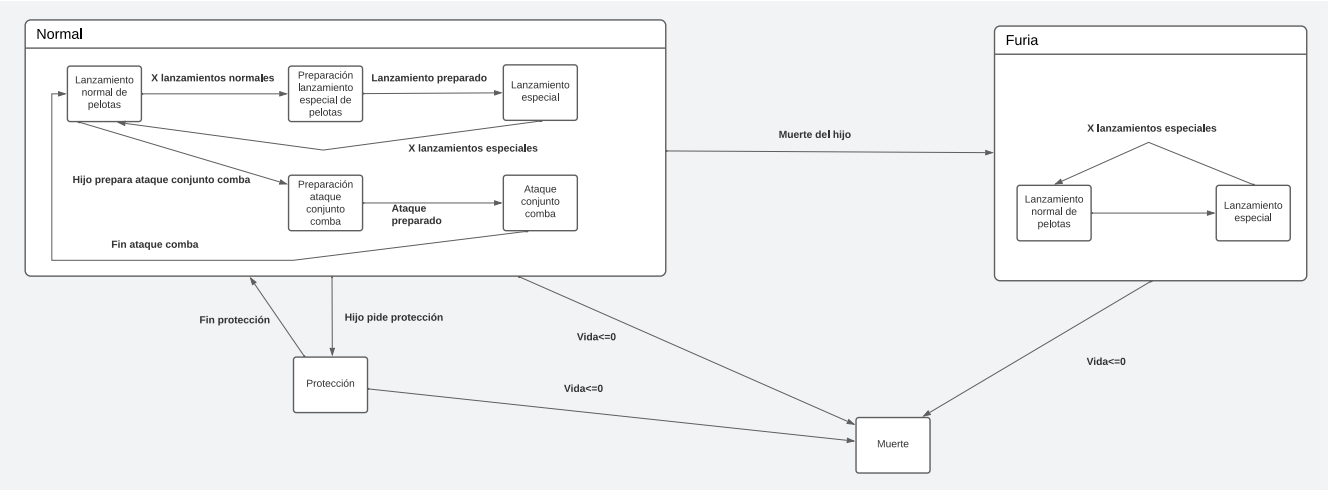
El comportamiento del enemigo a distancia es bastante similar al del enemigo a melee, exceptuando que además ha de comprobarse que cuando se encuentre el jugador dentro del rango de ataque no hayan obstáculos entre el enemigo y él (una trayectoria limpia).



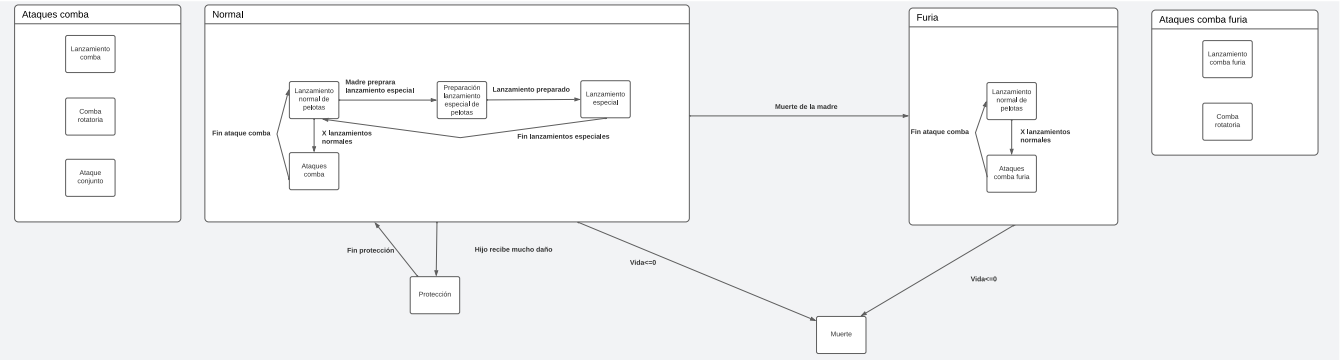
Máquina de estados del Nerd (jefe de la sección de electrónica)



Máquina de estados de la Madre e Hijo (jefes de la sección de juguetería)



La máquina de estados del Hijo sería la siguiente:



8. Elementos del juego

8.1. Armas

El jugador podrá encontrar dos tipos de armas: armas de ataque cuerpo a cuerpo (o melee) y armas de ataque a distancia. Dichas armas las usará el jugador para enfrentarse a los enemigos que encontrará a lo largo de la partida. Solo se podrá llevar dos armas, que serán utilizadas por el líder del grupo (el personaje controlado por el jugador).

En total habrá 8 armas en el juego, la mitad cuerpo a cuerpo y la otra mitad a distancia. Habrá dos armas genéricas que podrán encontrarse en cualquier nivel y dos armas específicas por nivel (6 en total).

Armas genéricas

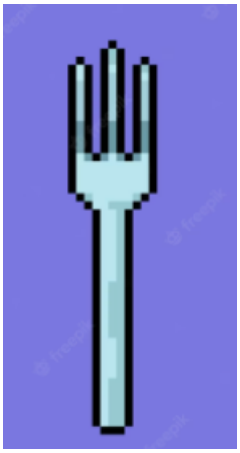
- Bate de beisbol: arma básica de ataque cuerpo a cuerpo.

- Bolsa de productos: Bolsa de la compra llena de productos, los cuales se podrán lanzar a los enemigos.

Armas temáticas

Alimenticia:

- Barra de pan: La dureza de este pan lo convierte en un arma perfecta para golpear a los enemigos



- Tenedores: Arma afilada y arrojadiza que nada tiene que envidiar a un cuchillo arrojadizo.

- Rodillo: Aunque su función original era la de amasar, su dureza y contundencia la convierten en un arma capaz de rivalizar con el bate de Beisbol.

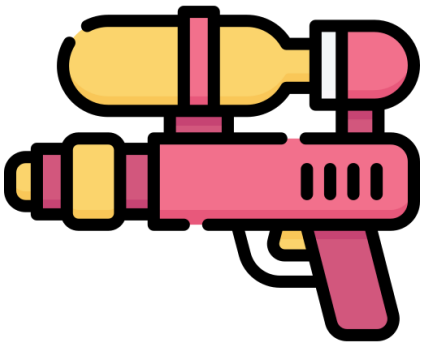
Electrónica:

-Ratón látigo: Los periféricos de ordenador pueden tener otros usos, en este caso, el ratón puede usarse como látigo, ofreciendo un buen alcance a la hora de golpear enemigos.

-Secador lanzallamas: Un secador defectuoso que se encontró en las rebajas. No es muy útil a la hora de secar el pelo, pero sus llamas pueden usarse para dañar enemigos.

Juguetería:

-Yoyó (se queda rodando en un punto y golpea consecutivamente al enemigo): Arma cuerpo a cuerpo de alcance parecido al ratón látigo. Al lanzarse, su extremo se queda girando en un punto, provocado daño consecutivo al enemigo.



-Pistola de agua: El chorro continuo de la pistola de agua, similar a un láser, provoca un gran daño a los enemigos que empapa. (funcionamiento similar a un láser)

8.2. Accesorios

Otro tipo de objeto que encontrará el jugador a lo largo de la partida serán los clásicos accesorios que otorgarán distintas bonificaciones a los personajes. Dichos accesorios podrán encontrarse en cualquiera de los niveles del juego, es decir, son genéricos y no específicos de cada sección.

A diferencia de las armas, que solo pueden ser llevadas por el personaje controlado por el jugador, los accesorios podrán ser llevados por el resto de personajes, aunque no sean el líder (es decir, no estén bajo el control del jugador en ese momento). Cada personaje podrá llevar un único accesorio y su efecto solo se encontrará activo cuando el personaje que lo lleva equipado sea el líder del grupo.

En total habrá cuatro accesorios:

- El casco: Protección básica para cualquier guerrero/comprador del Black Friday. Aumenta la vida del personaje.

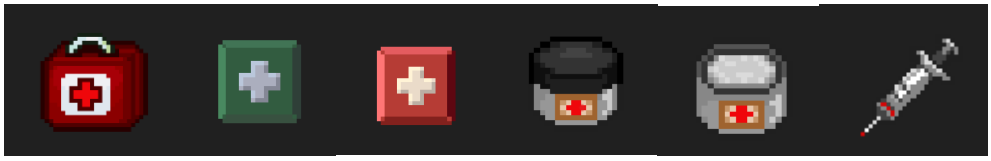
- Los guantes: El suave terciopelo de estos guantes, además de proveer cierto estilo, permite a su dueño atacar con más brio. Aumenta la fuerza del personaje.

- El parche de café: Pequeño parche de café que otorga un subidón de cafeína a su usuario. Aumenta el número máximo de usos de habilidad del personaje.

- Botas: Cómodo y estiloso calzado cuyo diseño aerodinámico mejora la capacidad de movimiento del portador. Aumenta la velocidad de movimiento del personaje.

8.3. Ítems consumibles

Existirán una serie de ítems consumibles que soltarán los enemigos al ser derrotados. Dichos ítems afectarán al líder del grupo al momento de recogerlos y desaparecerán de la sala cuando esta sea abandonada, junto con cualquier arma o accesorio que se haya dejado atrás (el conserje las elimina).



- Botiquines: Hay pequeño y grande. Sirve para recuperar vida

- Zumo/Refresco energético: Pequeño y grande. Sirve para recuperar usos de la habilidad

- Munición: Pequeño y grande. Como su nombre indica, sirve para rellenar munición del arma que se esté llevando en el momento de recogerla

8.4. Artículos:

El objetivo es reunir tres artículos clave de cada sección, los cuales sirven como llave para entrar a la sala del jefe. Los artículos por sección son los siguientes:

Sección de alimentación:

Una manzana

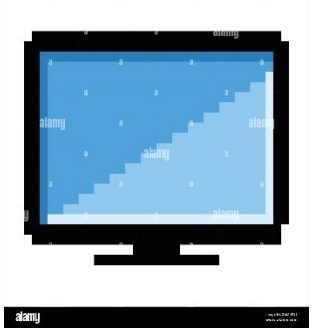
Un brick de leche

Galletas

Pizza

Hamburguesas

Sección electrónica:



Televisión

Móvil

Portátil

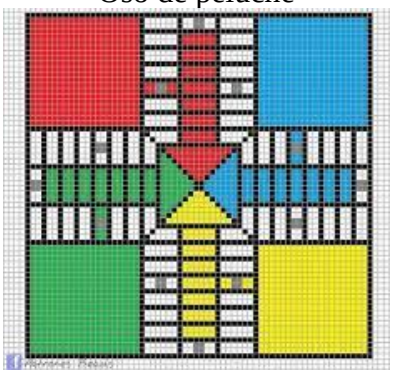
Lavadora

Videoconsola

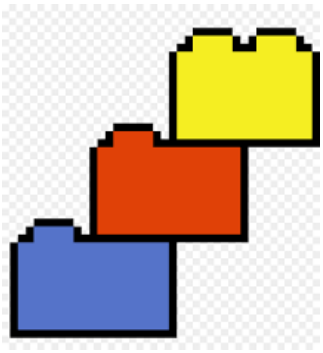
Sección de juguetería:

Oso de peluche

Cubos de letras



Juego de mesa



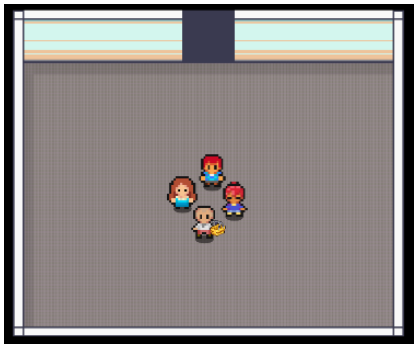
Caja de bloques de construcción

Triciclo

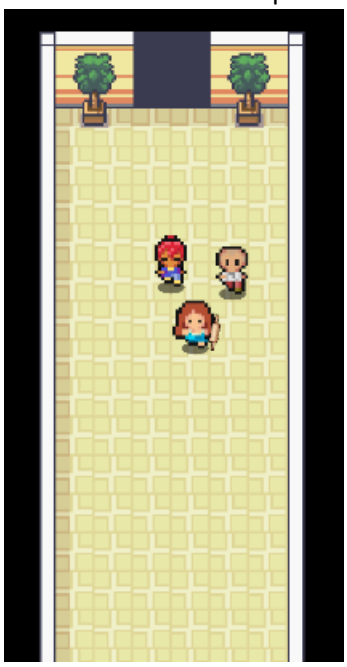
8.5. Escenario

Los niveles del juego se componen de una serie de salas que se conectan de forma aleatoria. Debemos destacar los siguientes tipos de salas:

- Sala inicial: Pequeña habitación vacía con una única puerta. Cada vez que se inicia el nivel la familia aparecen en la sala inicial, que podría definirse como la casilla de salida.

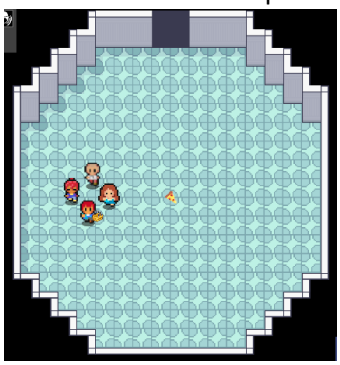


- Pasillo: Sala estrecha con suelos y paredes de color amarillo que solamente cuenta con dos puertas. Suele estar decoradas por macetas con plantas. No suele contener muchos enemigos.



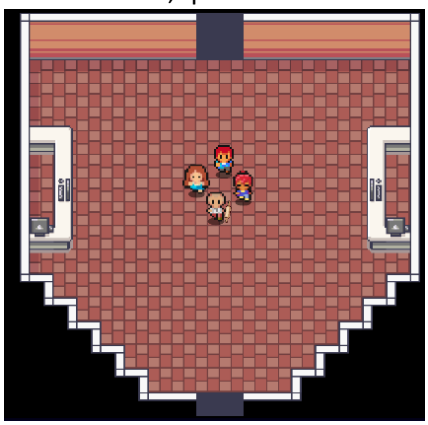
- Sala de almacenes: Sala normalmente espaciosa con suelos grises y paredes de color azul claro, con una cantidad variable de puertas. Puede tener una, tres o cuatro puertas. Suelen tener estanterías con productos u otros muebles. Suelen contener una gran cantidad de enemigos.

- Sala de artículo: Sala azul que contiene uno de los tres artículos necesarios para pasar el nivel. Solo cuenta con una única puerta.



- Sala del botín: Sala de trastero en la que pueden encontrarse cajas de botín, que contienen distintos accesorios. Solo cuenta con una puerta.

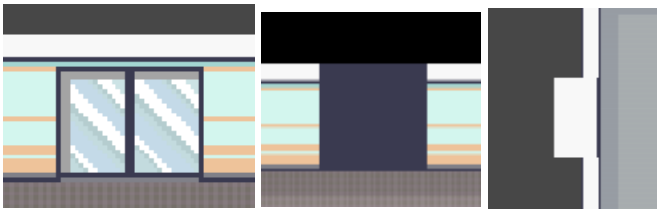
- Sala prejefe: Sala con azulejos y paredes de color rojizo, con dos mostradores a sus extremos. Solo cuenta con dos puertas, siendo la puerta del extremo superior, la entrada a la sala del jefe y del combate final, que solo se abrirá si el jugador ha reunido los tres artículos del nivel.



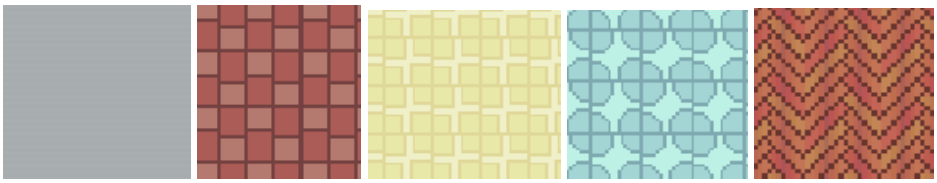
- Sala jefe: Sala con azulejos y paredes de color rojizo (como la sala prejefe). Solo cuenta con una única puerta, la puerta de entrada, que se cerrará de inmediato al entrar en la sala, dando comienzo a la batalla contra el jefe de nivel.

En cada sala podemos encontrar distintos elementos, de los que debemos destacar:

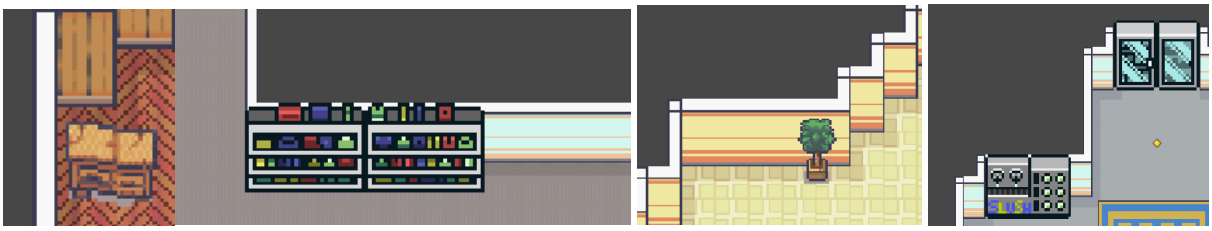
- Las puertas: Permiten trasladar al jugador a una nueva sala. Se cierran cuando hay enemigos en la sala, impidiendo al jugador salir.



- Suelos: Distintos tipos dependiendo de la sala.



- Paredes y obstáculos: Elementos estáticos que los personajes no pueden atravesar. Hay diversos tipos dependiendo de la sala.



- Cajas de botín: Cajas que solo se encuentran en las salas de botín. Permiten conseguir accesorios.

9. Interacción

9.1. Controles

Teclas	Función
W A S D	Movimiento del personaje
Mayús	Activar habilidad
Q	Cambiar líder
Ratón Izquierdo	Atacar
Rueda del ratón	Cambiar arma
E	Viajar entre salas
Escape	Abrir el menú de opciones
C	Coger accesorio/arma
X	Soltar accesorio

Con el movimiento del ratón, el punto de mira del personaje también se moverá.

9.2. Pantalla de Splash

La pantalla de Splash, que sirve como presentación al juego, consiste en un título que baja desde el extremo superior de la pantalla, los Sprites animados de la familia Remunson, que suben desde el extremo superior de la pantalla, y un pequeño texto intermitente que indica la necesidad de pulsar una tecla para continuar y abrir el menú.

Si pasan 50 segundos sin que el jugador toque ninguna tecla, el juego se iniciará solo y se abrirá el menú principal.

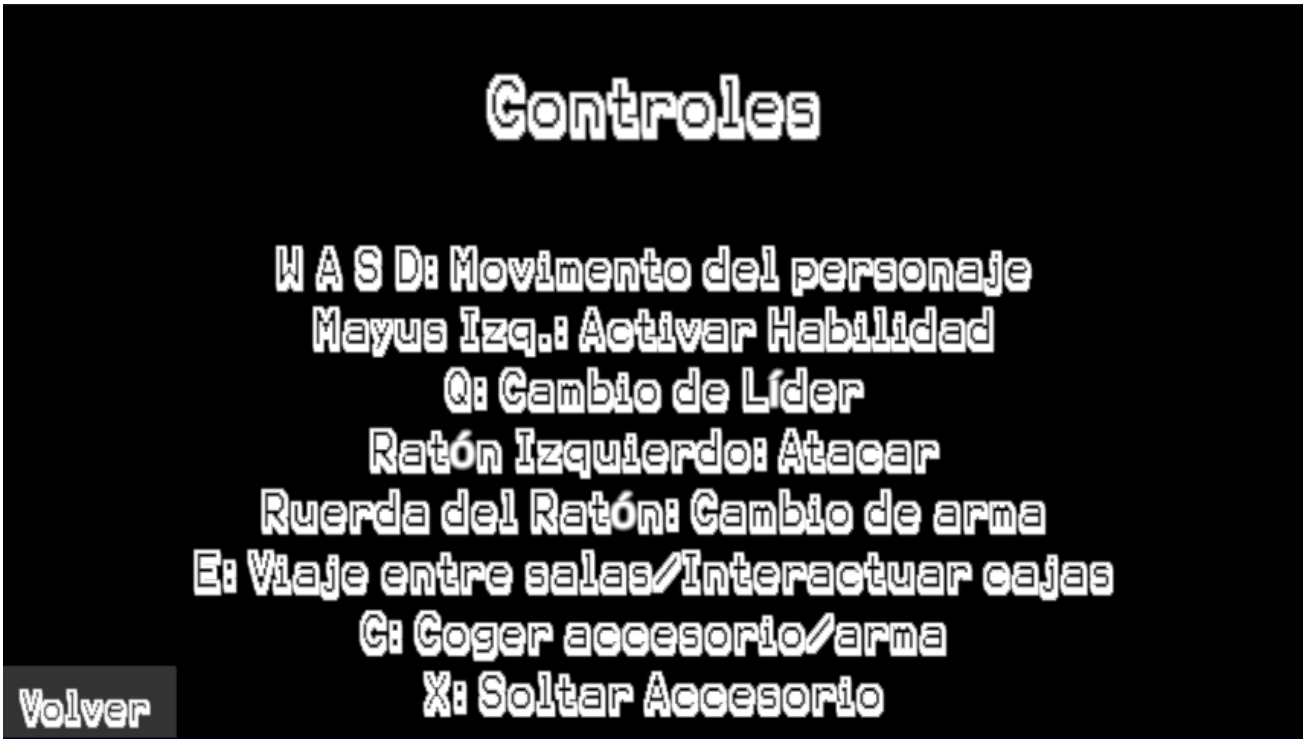
9.3. Menú principal

El menú principal del juego consiste en una imagen generada mediante inteligencia artificial que se ha retocado para visualizar mejor los botones y el título del juego sobre ella.

Cada botón representa una opción: el botón Jugar arranca el juego, el botón Salir finaliza el juego y el botón Opciones abre un panel con la configuración e información adicional del juego.

En dicho panel se puede encontrar el clásico slider para cambiar el volumen, un checkbox para activar y desactivar la pantalla completa y tres para conocer más datos acerca del juego: Controles, Historia y Créditos.

La opción Controles muestra los controles utilizados en el juego.



La opción Historia muestra un resumen del argumento del juego e información sobre los personajes principales.

En cuanto a la opción de Créditos, al pulsar sobre ella se muestran los nombres de los integrantes de nuestro grupo.

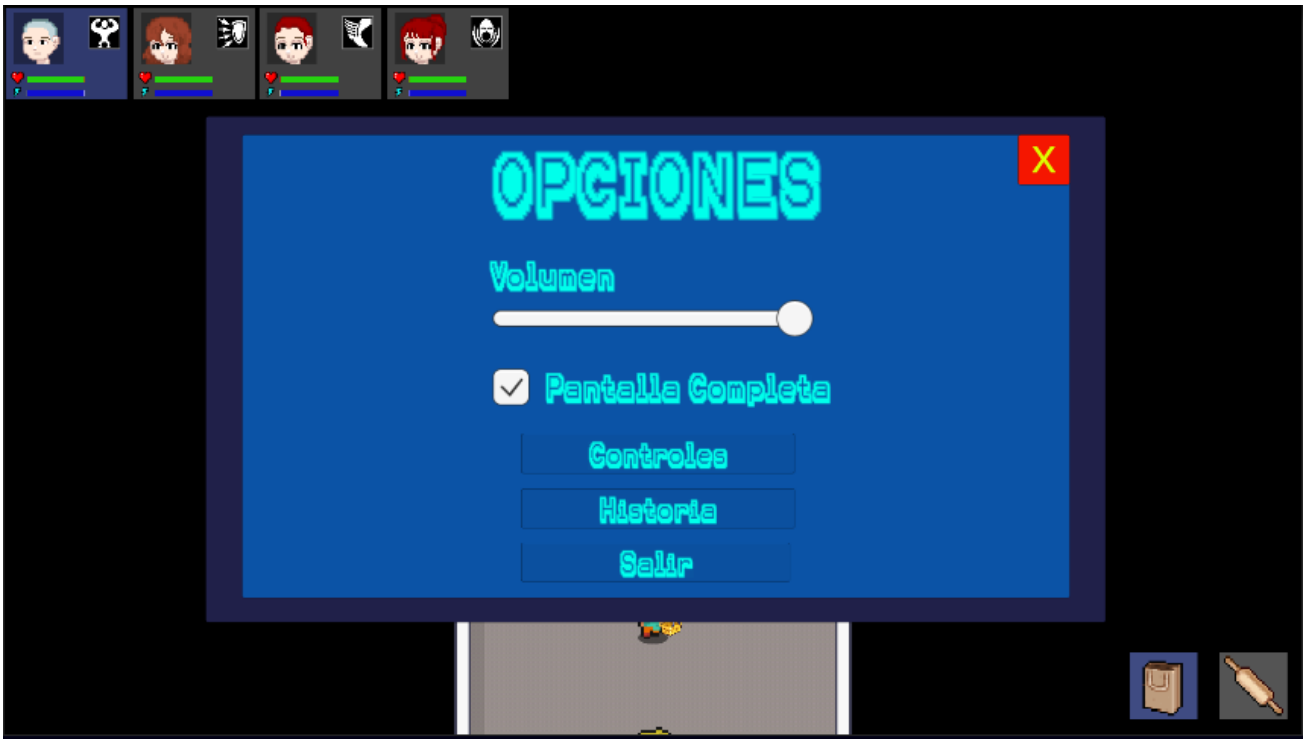
9.4. HUD

La HUD del juego durante la partida no es muy compleja. En la parte superior de la pantalla aparece la información de los personajes (vida, usos de habilidad, habilidad del personaje y el accesorio equipado). La tarjeta del líder es resaltada en azul y las tarjetas de los personajes caídos son tachadas.

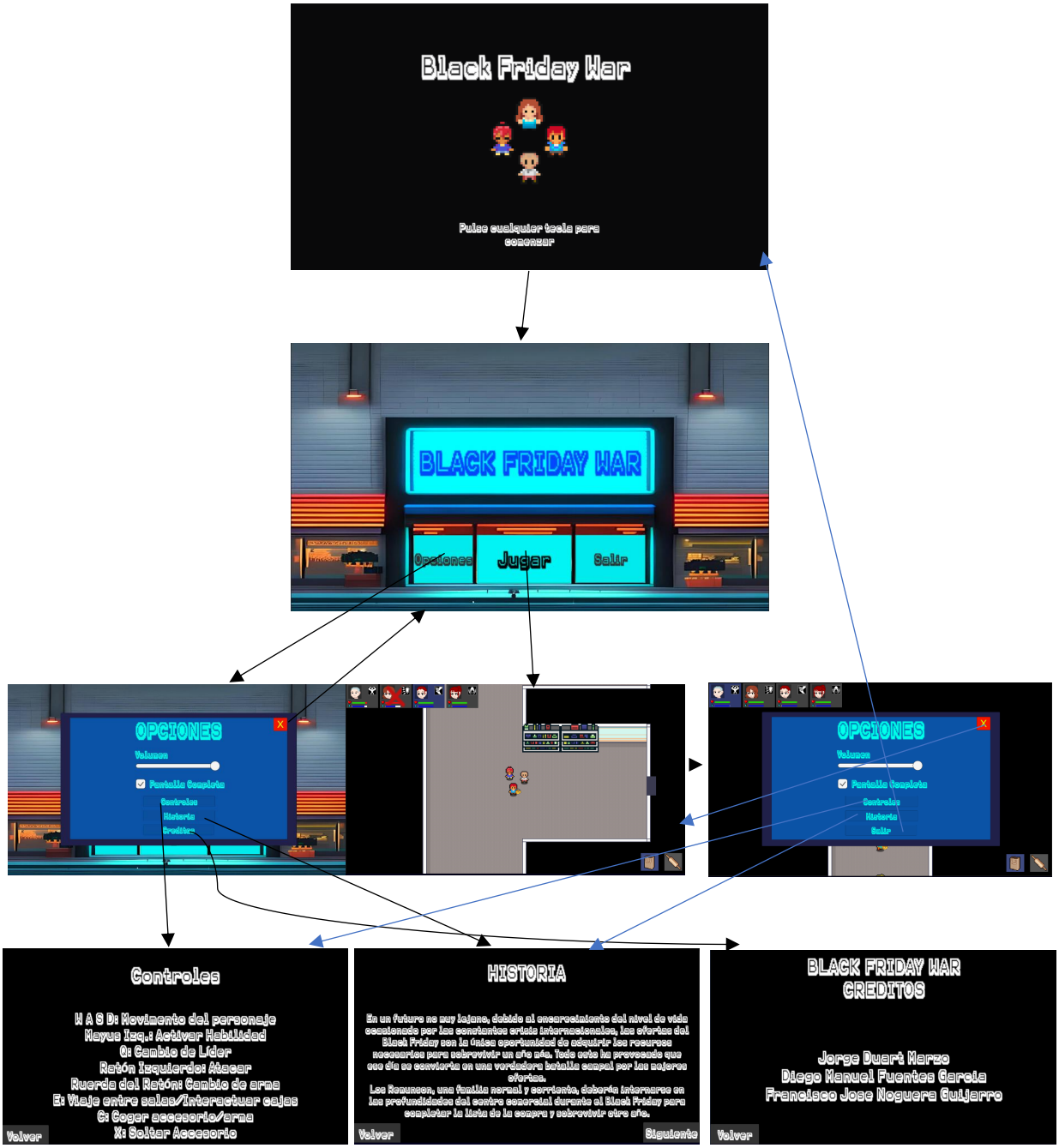
En la esquina inferior derecha, se indican las armas equipadas por el jugador, siendo la activa la resaltada con un color azulado.

9.5. Menú de pausa

Durante la partida, al pulsar la tecla ESCAPE se abre el clásico menú de pausa, parando la acción del juego. En nuestro caso, el menú de pausa Contiene las mismas opciones que el panel de opciones del menú principal, exceptuando la opción de créditos, que se sustituye por la opción de Salir, que finaliza la partida y devuelve al jugador a la pantalla de Splash.



9.6. Diagrama de navegación



10. Repositorio GitHub

Enlace

diego-manuel1/BlackFridayWarProject (github.com)

Equivalencia usuarios/miembros del equipo

diego-manuel1 – Diego Manuel Fuentes García

FranNogui – Francisco Noguera Guijarro

DuiMarz – Jorge Duart Marzo

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Glosario

- AI o IA: Inteligencia Artificial
- GDD: Game Design Document (Documento de diseño de juego)
- NPC: Non Playable Character (Personaje no jugador)
- ODS: Objetivos de Desarrollo Sostenible
- TFG: Trabajo de Fin de Grado
- UPV: Universidad Politécnica de Valencia
- BT: Behavior Tree (Árbol de comportamiento)
- UT: Unidad de Trabajo
- PA: Prueba de Aceptación
- UI: Interfaz de Ususario