Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo de Fin de Grado

Grado en Ingeniería Informática
Mención en Ingeniería del Software

Desarrollo de un videojuego tipo “Tower Defense” para realidad virtual basado en el Castillo de la Mota

Autor:
D. Julio Escudero Cuesta

Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo de Fin de Grado

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

Desarrollo de un videojuego tipo “Tower Defense” para realidad virtual basado en el Castillo de la Mota

Autor:
D. Julio Escudero Cuesta
Tutor:
D. Carlos Enrique Vivaracho Pascual

Agradecimientos

Quiero agradecer a mi familia y amigos más cercanos el apoyo que me han brindado durante estos últimos meses de trabajo y durante toda mi etapa en la universidad. Sin su cercanía todo esto hubiera sido más difícil, así que solo puedo decir darles las gracias.

Resumen

El sector de los videojuegos ha crecido drásticamente en los últimos años. Los últimos datos indican que los beneficios y el valor de mercado de este sector supera incluso al cine y a la televisión. Además, las inversiones se han multiplicado y las investigaciones más recientes apoyan la idea de utilizar los videojuegos no solo como pasatiempo, sino como herramienta educativa.

Dentro de este ámbito, la realidad virtual está siendo demandada como líder de la próxima generación. Aunque las principales empresas de desarrollo de videojuegos no destinan aún sus principales recursos a desarrollar aplicaciones para dispositivos de realidad virtual, si es cierto que cada vez son más y más los videojuegos que salen al mercado para esta plataforma y la tecnología detrás de ella también avanza a pasos agigantados.

En este proyecto se persigue diseñar un videojuego de estrategia para realidad virtual en el que el jugador tenga que defender una réplica del Castillo de la Mota de Medina del Campo del asedio de los enemigos, todo ello con armas y tecnología propias de la época de inicio de la Edad Moderna. Con esto se promueve el sentimiento de defensa de lo personal del ser humano y sumado al realismo que proporciona esta tecnología y la ambientación del videojuego se consigue que el jugador se sienta como un auténtico comandante dirigiendo y coordinando sus tropas. Además, el juego pretende enseñar conceptos históricos de la Edad Moderna y promover el turismo basado en videojuegos

Abstract

Gaming industry has grown drastically over the last few years. Last data show that profits and market value in this sector surpass even the cinema and tv industry. Furthermore, investment have increased greatly and latest researches support the idea of using video games not only as a hobby, but also as a learning tool.

Within this scope, virtual reality is being demanded as the leader of the next generation. Even though main game development companies do not allocate their main resources to develop applications for these virtual reality devices yet, it is true that more and more video games for this platform get into the market and the technology that lies beneath it is also improving by leaps and bounds.

In this project we attend to design a strategy video game for virtual reality in which the player has to defend the Castillo de la Mota in Medina del Campo from the siege of the enemies, all of these with weapons and technology from the Modern Age era. With all of these the game promotes the sense of human beings of defending their privacy and combined with the realism that this technology brings and the video game environment we make the player feel as if they were a real commander leading and coordinating their troops. Moreover, the game intends to teach historical concepts of the Modern Age and impulse the turism based on video games.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

1.1. Contextualización

El sector de los videojuegos está en alza y ha evolucionado de manera exponencial en las dos últimas décadas. Han ido instalándose en la vida diaria y poco a poco se ha normalizado su uso no sólo para fines puramente lúdicos, sino también con fines educativos y de aprendizaje y con pinceladas de competitividad real. Para tener una cifra de referencia, el número de videojuegos para ordenador lanzados al mercado en 2021 de la mano de la plataforma *Steam* fue de 10.696 (Figura 1), 1000 más que el año anterior, y esto solo se corresponde con una plataforma de videojuegos y una consola.[1]

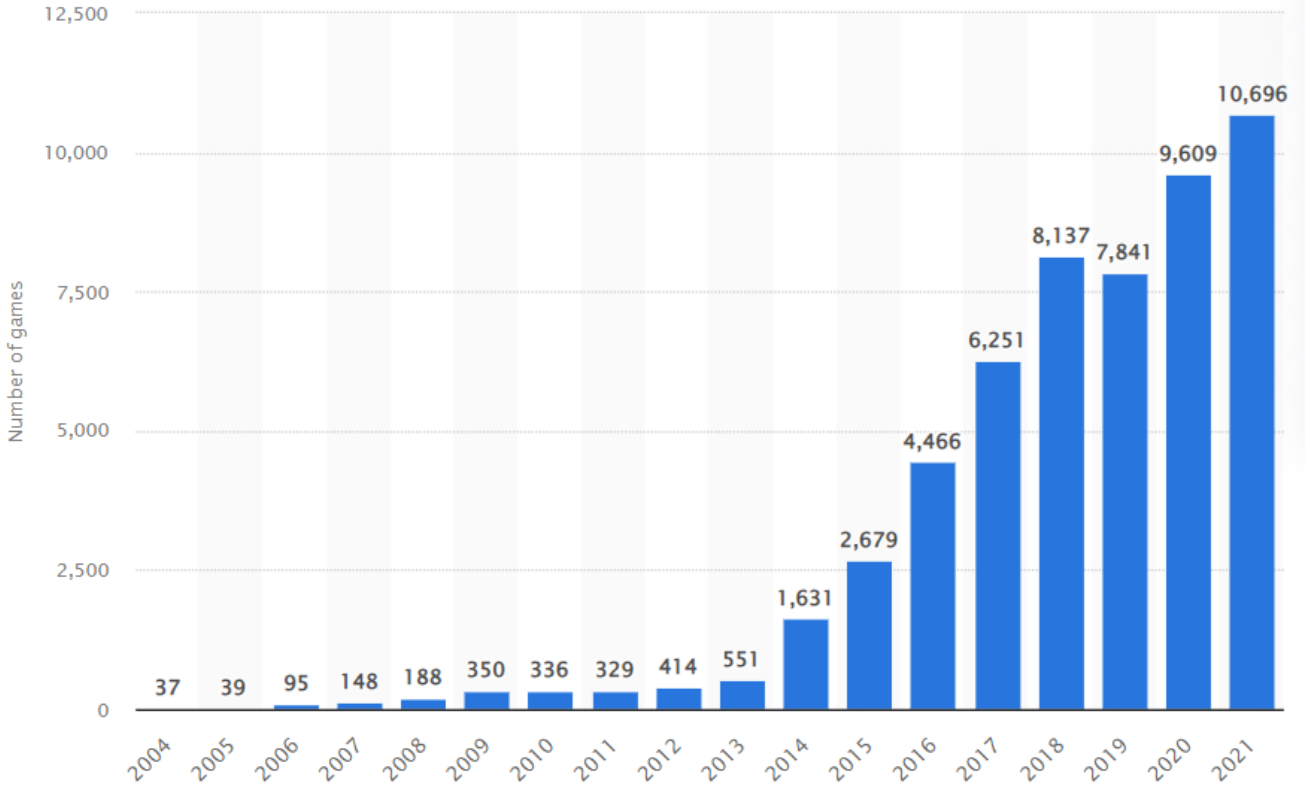


Figura 1: Diagrama de número de juegos lanzados en Steam cada año

Establecer un número concreto de videojuegos desarrollados en 2021 es una tarea ardua. Aunque sí sea factible recoger estadísticas del número de videojuegos publicados según las plataforma más conocidas o según la compañía desarrolladora más populares o con más renombre, existen pequeños vaivenes en relación a la duración que puede tener un juego en la plataforma, videojuegos que no completan su desarrollo o videojuegos que se desarrollan para ser jugados de manera gratuita y sin ningún coste de compra dentro del propio juego. El único dato que se puede obtener de manera muy aproximada es el de un total de 2.500 millones de juegos solo en Estados Unidos. Una cifra que quizás por sí sola no sea tan asombrosa, pero que si se escribe junto a un valor de mercado estimado de 178.730 mil millones en 2021, y junto a una estimación de hasta 268.810 mil millones en 2025, impresiona (Figura 2). [2]

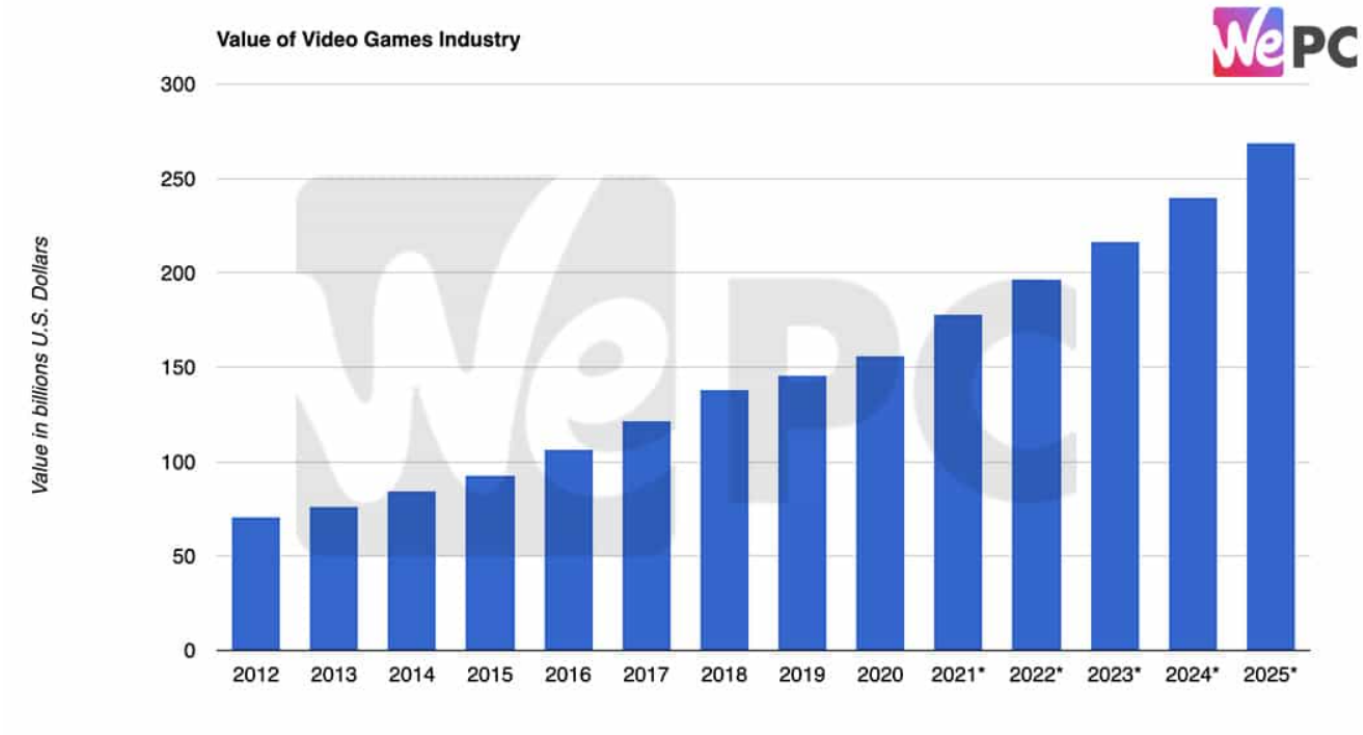


Figura 2: Diagrama valor de mercado industria videojuego estimado por año

1.2. Motivación

El principal reto de este proyecto es la puesta en escena de la tríada videojuegos, realidad virtual e historia. La combinación de las dinámicas y la acción que traen consigo los videojuegos, la inmersión y el realismo (valga la redundancia) que aporta utilizar la realidad virtual y sumarle a todo ello el aprendizaje histórico, puede resultar en un conjunto de experiencias muy enriquecedoras para el usuario.

A esto se le añade que es un proyecto de mayor envergadura comparado con lo que podría considerarse un TFG normal, siendo un equipo compuesto por tres personas, cada una especializada en su campo. Las otras dos personas han reunido toda la documentación histórica necesaria para el juego, han preparado el escenario donde se va a desarrollar y han diseñado los menús. Por otra parte, mi trabajo ha sido diseñar y programar la lógica del videojuego.

Centrando más la atención en el mundo de los videojuegos, se persigue obtener una jugabilidad realista de la defensa de un castillo de unos invasores, poniendo en práctica conceptos de estrategia y organización, a la par que la administración de recursos disponibles, tanto militares como monetarios. Con todo ello se pretende que el jugador adopte el papel de general de defensa y que poniéndose en la piel del mismo, tome las mejores decisiones posibles para defender el castillo el mayor tiempo posible.

Por todo ello, se pretende que el juego sea atractivo para personas que tengan cierta familiaridad con los videojuegos y más en concreto, con aquellos que impliquen cierta estrategia, aunque no por ello se excluye para otros tipos de perfiles, desde jugadores de juegos de otra índole o incluso personas que no dediquen apenas tiempo en los videojuegos pero que sientan curiosidad y atracción por el realismo proporcionado tanto por la realidad virtual como por la fidelidad al apartado histórico.

Este proyecto ha sido propuesto por la empresa Irzón SA a partir de la digitalización 3D del Castillo de la Mota y supone un trabajo paralelo al proyecto de D. Alfredo Fernández Ramos “Desarrollo de videojuegos para la promoción del castillo de la Mota utilizando tecnología de realidad virtual”[3]. A diferencia de este proyecto que era un videojuego de disparos en primera persona, el videojuego que se desarrolla aquí tiene una perspectiva top-down y entra dentro de la categoría de *Tower Defense.*

1.3. Objetivos

El objetivo general del TFG es desarrollar un videojuego *Tower Defense* de realidad virtual utilizando como escenario una recreación en 3D del Castillo de la Mota que fusione la diversión del propio videojuego con la promoción del castillo y educación histórica.

Este objetivo general se divide en los siguientes subobjetivos:

- Reunir documentación histórica para la ambientación del videojuego
- Analizar videojuegos conocidos con la misma tecnología y/o del mismo género para entender las características de su éxito.
- Entender cómo funciona un modelo 3D
- Desarrollar y construir un modelo GDD (Game Design Document) junto con el resto de miembros del equipo [Apéndice I]
- Implementar las diferentes programas y servicios para que el videojuego pueda funcionar en unas gafas de realidad virtual
- Diseñar y programar la lógica del videojuego recogida en el GDD

1.4. Organización de la memoria

A continuación se exponen los capítulos en los que se divide la memoria y una breve descripción de cada uno de ellos.

Teoría de los videojuegos

En este primer apartado se explicará y definirá el concepto teórico de videojuego y de aquellos elementos que forman parte de él. Además se explicará con detalle el modelo MDA y las etapas de desarrollo de un videojuego estándar.

Tecnología de la Realidad Virtual

Para continuar, se explicará de forma breve en qué consiste y cómo surge la realidad virtual y se hablará en concreto sobre las prestaciones que ofrecen las gafas de realidad virtual Oculus Quest 2, que son las que se usan en el proyecto

Juegos relacionados

En este apartado se nombrarán y definirán brevemente los mejores juegos pertenecientes al género *Tower Defense* y también los mejores juegos para Realidad Virtual hasta la fecha. Se analizará su éxito desde un punto de vista técnico.

¿Qué es Unity?

A continuación se hablará de qué es Unity como programa de desarrollo software, su historia y sus características principales y las diferentes herramientas que ofrece. Además se expondrán las opciones de adquisición de este software disponibles actualmente.

Entorno de desarrollo Rider

Después se incluirá un pequeño apartado en el que se explicará el origen del IDE que se va a usar para el desarrollo del proyecto y se realizará una comparación técnica con los otros entornos de desarrollo disponibles para Unity

Planificación del proyecto

A continuación se pasará a explicar los factores tenidos en cuenta para la planificación del proyecto y su desarrollo. Se realizará un análisis de los riesgos asociados al proyecto, el coste previsto teniendo en cuenta los elementos que se van a utilizar y por último una explicación de la metodología escogida para el desarrollo y un repaso por las diferentes fases del mismo.
Conclusiones

En este apartado antes de finalizar se expondrán las conclusiones obtenidas tras la realización del proyecto.

Desarrollo del videojuego

En este apartado se exponen las características principales del videojuego que se desarrolla y el por qué de que se quieran incluir en el juego y además se expone cómo se espera que el jugador reaccione a la información que recibe. Además se incluye una primera versión del Documento de Diseño del Videojuego.

Conclusiones

En las conclusiones finales se recoge un resumen de lo que ha supuesto embarcarse en el desarrollo de este proyecto, qué errores se han cometido y cómo pueden solucionarse y se expone los conocimientos adquiridos a lo largo del periodo de trabajo.

Trabajos futuros

Por último, se nombran diferentes propuestas para mejorar tanto la jugabilidad en el videojuego como el software desarrollado.

2. Teoría de videojuegos

En cualquier disciplina de ingeniería o más bien, del desarrollo de un producto desde cero, existe una trayectoria de evolución desde su origen hasta la actualidad en la que se pueden observar ciertos patrones a seguir o buenas prácticas que ayudan o que sirven de guía en este camino tan abstracto compuesto por ideas, ambiciones, ilusión y mucho mucho esfuerzo.

Con los videojuegos no es ni mucho menos diferente y aunque es algo relativamente contemporáneo, existen ciertas personas que se han dedicado a establecer modelos o patrones para establecer definiciones formales de videojuego y su correcto desarrollo.

2.1. Historia y Origen

Para intentar entender el éxito que tienen en la actualidad los videojuegos es imprescindible echar un vistazo mucho más atrás e indagar en el origen del juego en la civilización, no en vano, este es el predecesor del videojuego.

Para ello, hay que remontarse a más de 2.000 años atrás, con las primeras civilizaciones que surgieron en la tierra, porque ya en la antigua Mesopotamia, Egipto, Grecia, China, etc, se practicaba el juego como una actividad lúdica. No únicamente juegos de mesa como el Ajedrez (Figura 3) o el Mahjong y todas sus variantes o versiones, sino también competiciones deportivas o de carreras o todo tipo de luchas. [4]

Figura 3: Pintura de la Edad Media de una partida de ajedrez

Y esto ha ido evolucionando a lo largo de los siglos con juegos como los dados, la ruleta o el trile, los cuales alcanzaron una popularidad enorme durante la Edad Media, o juegos de combate o escaramuzas, los toros y por supuesto juegos sencillos de pelota. Muchos de estos juegos se siguen practicando incluso a día de hoy y han ido perfeccionándose o variando para resultar más divertidos para el jugador como por ejemplo la rana o los bolos, haciendo mención especial a todos los deportes de pelota. [5]

Será a partir de mediados del siglo XX cuando surjan los primeros juegos con interacción persona-máquina. Resulta difícil calificar cuál de todos estos se podría considerar como primer videojuego, ya que no existe una definición formal y consensuada para este concepto, por ello se expondrán diferentes candidatos sin afirmar con plena confianza que uno u otro sea el primerizo.

En 1950, se presentó en la Exposición Nacional Canadiense un computador llamado Bertie the Brain (Figura 4) capaz de jugar al 3 en raya contra un ser humano, con dificultad ajustable y que se construyó con el objetivo de mostrar en la feria el pequeño tubo de vacío que se había utilizado para su construcción. Disponía de un teclado electrónico donde el jugador seleccionada su jugada y esta se reproducía en un panel superior mucho más grande y vistoso. Sin embargo, la máquina fue desmontada y posteriormente olvidada, ya que al poco tiempo este componente electrónico quedó obsoleto por el surgimiento de los transistores en ordenadores. [6]

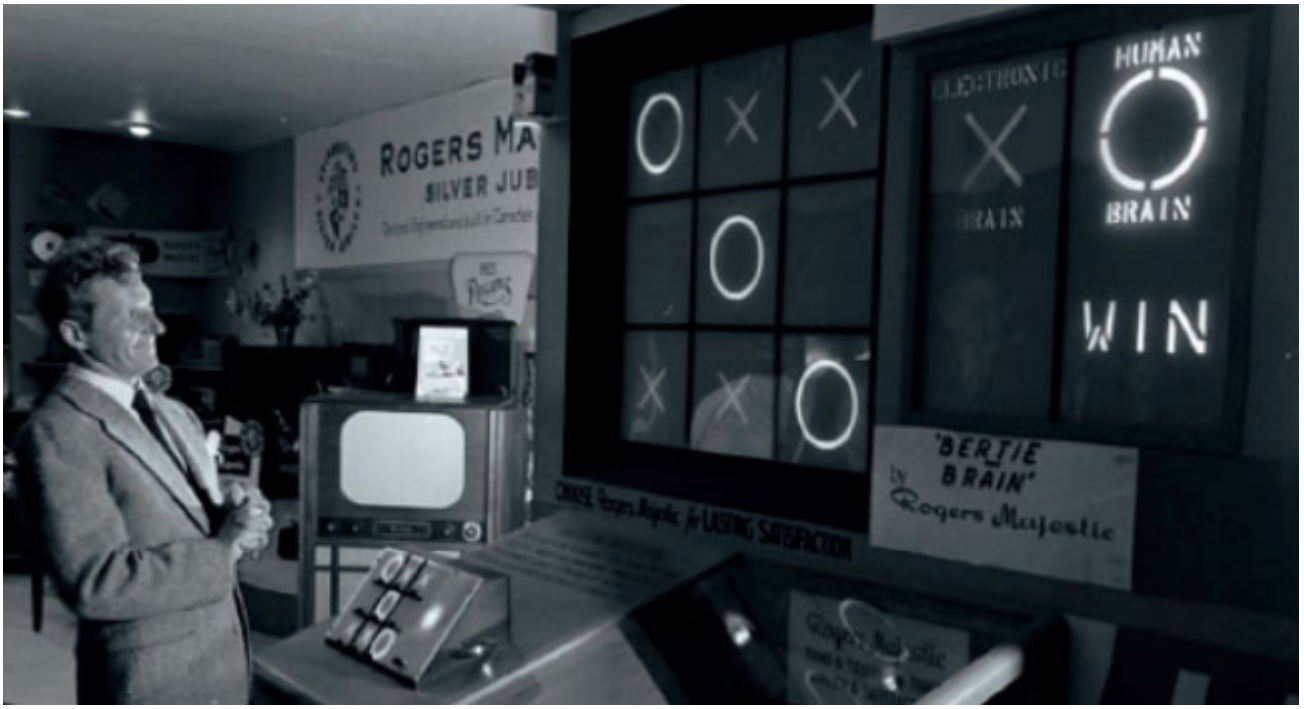


Figura 4: Bertie the Brain

En 1951, se expuso en una feria británica de la mano de la empresa Ferranti, fabricante de equipos electrónicos, un enorme computador capaz de jugar al nim, un juego matemático de estrategia para dos jugadores, tanto a su forma tradicional, como a la forma “invertida”. Dicho ordenador estaba inspirado en una máquina electromecánica expuesta por primera vez en 1940 que también era capaz de jugar al nim. Este computador también fue desmontado posteriormente para usar sus piezas en proyectos de ámbito más científico.

Otra fecha relevante en la historia de los videojuegos es 1952, cuando aparecieron *OXO*, otra versión del juego 3 en raya muy simple y *Draughts,* versión electrónica de las damas, la cuál en su origen era más bien lenta y pesada de jugar en contra, pero que tras progresivos refinamientos, en 1962 se consiguió que derrotara a campeones estadounidenses del juego. [7]

Un proyecto que es también imprescindible destacar es *Tennis for Two* (Figura 5) , un juego de tenis construido utilizando una circuitería de transistores y usando como pantalla un osciloscopio. Unas líneas que representaban las raquetas eran manejadas con dos controladores que se habían construido para dicho fin. A pesar del gran impacto que causó entre aquellos que probaron esta máquina, de nuevo el proyecto fue desmantelado para usar sus piezas para otros fines.

Figura 5: Tennis for two

Todos estos proyectos mencionados hasta ahora no tuvieron la ocasión de ver la luz y se mantuvieron en talleres y laboratorios científicos, además de que el principal motivo de su desarrollo era académico y no lúdico. No sería hasta 1961 cuando unos estudiantes del MIT desarrollaron *Spacewar!*, un juego espacial de disparos para dos personas que ocupaba 9k de memoria y que ha sido precedente de muchos otros juegos del estilo a lo largo de la historia. Ya en 1971 aparecieron *Galaxy Game* y *Computer Space*, dos videojuegos del mismo género que corrían en sus propias consolas y que usaban un sistema de monedas para funcionar. Además en 1972 surge el videojuego *Pong* (Figura 6) , una versión electrónica jugable del ping–pong y que sirvió de impulsor del negocio del entretenimiento y del videojuego.

Como ya se ha expuesto antes, existe controversia a la hora de determinar cuál fue el primer videojuego de la historia. Sin embargo, sí se puede afirmar con rotundidad que estos últimos prototipos de los años 60 e inicios del 70 sentaron las bases de la industria moderna y pueden considerarse los padres de este sector. Es a partir de entonces cuando los videojuegos comenzaron poco a poco a evolucionar hacia algo más normalizado y cotidiano, debido a que fueron deshaciéndose de elementos mecánicos y estrambóticos. [8]

Figura 6: Pong

2.2. Definición de juego

En paralelo al surgimiento de los primeros juegos, diferentes figuras y personalidades del mundo antiguo establecen descripciones y ventajas de está práctica recreativa. Uno de los primeros autores que escribió sobre el juego fue Platón, valorando la utilidad práctica que podía tener en el proceso educativo de los jóvenes y que ayudaba al desarrollo de sus habilidades y carácter. Su discípulo Aristóteles exponía una opinión similar a la de su predecesor, aunque añadía que el juego compensaba la fatiga producida por el trabajo y que, por ende, tenía un efecto reparador y placentero en el ser humano.

En el siglo I, el abogado, retórico y pedagogo Marco Fabio Quintiliano, consideraba el juego como un elemento rompedor de la monotonía de la enseñanza y su uso educativo para evitar el cansancio temprano de los niños a los que enseñaba. Lo usaba también como un elemento motivador y que estimulaba el deseo de aprendizaje del alumno.

En el siglo XVII esté pensamiento pedagógico del juego se vuelve a recuperar y se populariza enormemente. Ideólogos como Rousseau o Fröebel plantean el juego como un instrumento fundamental en la educación y comienzan a surgir las primeras escuelas pedagógicas y también gran variedad de juguetes. Sin embargo no será hasta el siglo XIX cuando empiecen a surgir los primeros estudios propiamente dichos centrados en el juego y su definición y características.

Diferentes aproximaciones se llevaron a cabo. Herbert Spencer defendía que el juego era un exceso de energía propio de los niños que conservaban debido a su ausencia en la participación de actividades de supervivencia. Con ello el juiego comenzó a derivar de una actividad exclusivamente lúdica a algo

con lo que aprender y desarrollar capacidades motrices y/o estratégicas de preparación para la vida adulta, como defendía Karl Gross, que definía el juego como una forma de imitar los roles de una vida adulta y que permitía adaptar actividades que podían resultar tediosas para los jóvenes en interesantes.

En el siglo XX, numerosos psicólogos, pedagogos, filósofos e ideólogos establecieron aproximaciones y definiciones del juego. Muchos de ellos coincidían en que el juego tenía un carácter social, que proporcionaba placer, que combinaba capacidades motoras e intelectuales y que favorecía el desarrollo de la creatividad y de la propia construcción de la personalidad del ser humano. También opinaban que el juego brindaba la posibilidad de un aprendizaje lúdico y que fomentaba la competencia y suponía un reto a la inteligencia del jugador, además de que contenía elementos más técnicos como una serie de reglas previamente consensuadas y aceptadas por todos los jugadores y que jugar suponía liberarse de un exceso de energía.

Ya en las últimas décadas del siglo XX y en los primeros años del XI, diferentes escritores de ciencia ficción, profesores de universidad y sobre todo, diseñadores de videojuegos, recogieron en diferentes libros los fundamentos necesarios para la creación y concepción de juegos. En su mayoría coincidían en que el juego tenía una serie de objetivos y conflictos y que estaba gobernado por unas normas. Además, debía proporcionar retos a los jugadores y normalmente tenía un ganador o ganadores y un perdedor o perdedores. Por otro lado, el juego debía recompensar a los jugadores por sus logros y estos logros debían de tener un valor significante en el propio juego y no fuera de él y además, se debía garantizar que el acto de jugar era seguro y confiable para el jugador, que era algo artificial y que se llevaba a cabo en un ambiente fuera del mundo real. Algunos también defendían que el juego debía enganchar al jugador y que debía haber interacción entre los jugadores. [9]

2.3. Aproximación MDA

En cualquier materia o disciplina en la que se cree un objeto, construcción, prototipo o un producto tangible, sin importar el ámbito de su aplicación o su objetivo máximo, siempre se hacen uso de guías y metodologías de diseño y desarrollo que impulsar ese proceso creativo intrínseco en el trabajo de llevar a cabo un proyecto y de asegurar no solo de que finaliza, sino de que lo hace cumpliendo unos estándares de calidad.

El esquema de desarrollo de Mecánicas, Dinámicas y Estéticas (de aquí en adelante framework MDA) (Figura 7) , es un modelo de desarrollo de videojuegos propuesto por Robin Hunicke, Marc LeBlanc y Robert Zubek en la Conferencia de Desarrolladores de Videojuegos, San Jose, 2004, que presentaba una ayuda para los desarrolladores a la hora de comprender el proceso iterativo del desarrollo de un videojuego rompiendo la distancia entre el desarrollo y el diseño del videojuego a nivel algorítmico y programático, los objetivos o metas alterables del juego y las sensaciones o la experiencia que se quería transmitir al jugador final. [10]

Surge principalmente apoyado en la importancia del proceso iterativo connatural del diseño de software que consistía en descomponer el resultado final para refinar la implementación y, a su vez, analizar la implementación para depurar el diseño final, y busca proponer una solución al rompecabezas constante de que, a pesar de que cada individuo tenga como trabajo realizar tareas en un determinado campo de acción, en algún momento va a necesitar echar un vistazo u obtener información sobre lo que está ocurriendo fuera de su área de desempleo. Además, toda acción que se lleve a cabo en cualquier nivel del desarrollo va o bien influir en el resultado final si hablamos de una etapa primeriza, o bien influenciar en estas etapas tempranas si se tienen en cuenta decisiones como objetivos del juego o resultado deseado.

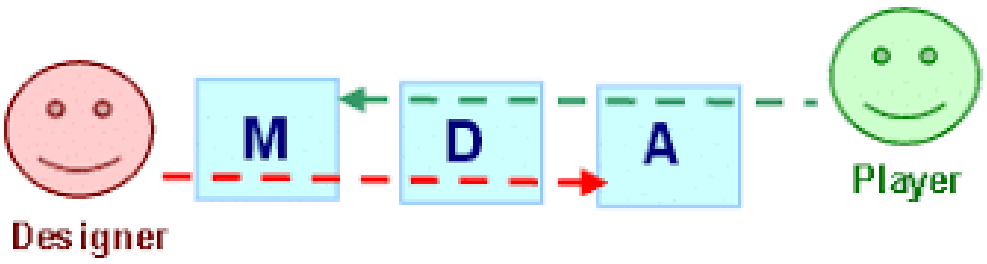


Figura 7: Modelo MDA

El modelo MDA proporciona una solución robusta a la dificultad de esta transición hacia ambas direcciones. Un videojuego es, al igual que cualquier otro producto, uno que se diseña y construye para que el usuario que lo compre obtenga un beneficio al hacerlo, ya sea utilidad o simple diversión. Sin embargo, la diferencia entre un videojuego y el resto es que el proceso de uso que lleva a cabo el jugador es ciertamente incontrolable e imprevisible. El resultado final es indudable, que el jugador disfrute y se divierta, pero las acciones que realice para conseguir dicho fin no son ni mucho menos exactas.

Una vez dicho esto, se puede mostrar que el framework MDA descompone el videojuego en tres distintos componentes: reglas, sistema y diversión del jugador, siendo cada una consecuencia o resultado de la anterior. Valiéndose de ellas como referencia, se formula su equivalencia en el diseño: **mecánicas, dinámicas y estéticas.**

2.3.1. Mecánicas

Por un lado, las mecánicas comprenden todo aquel componente puramente software del juego, desde el diseño hasta toda la parte de datos y algoritmos implementados. Hace referencia a las “reglas” del juego y conforman el conjunto de acciones y comportamientos que puede adoptar el jugador y por tanto, el alcance de control que tiene. En un videojuego de disparos algunas serían las armas, el movimiento del personaje, el punto de reaparición de los enemigos, etc.

Las mecánicas definen y ajustan el comportamiento de las dinámicas. Un ejemplo muy sencillo y a la vez muy ilustrativo que aparece en el artículo mencionado en el primer subapartado expone esta interacción. En el Monopoly, existe un gran problema de que, llegado un punto del juego, es muy predecible quién va a ganar la partida y quién la va a perder. No obstante, el periodo que sucede desde que los jugadores entreven el resultado hasta que se produce el final de la partida puede resultar largo y tedioso, y elimina toda la tensión de la partida.

Por ello expone propuestas como establecer ayudas para jugadores pobres e impuestos para los ricos, lo cuál ayuda a mantener las ganas de jugar en jugadores que estén perdiendo. También se proponen otras mecánicas para acelerar el juego, como establecer un impuesto para todos los jugadores de acuerdo al dinero que posean en cada momento, promoviendo que se gaste el dinero, o doblar el precio de todos los pagos de propiedades…

Es evidente que, tanto estas nuevas mecánicas propuestas como cualquier otra que pueda tener el juego deben ser probadas y refinadas de manera iterativa para garantizar que el juego tenga un balance real y no sea ni aburrido ni frustrante para el jugador. Por otro lado, es importante que las mecánicas introducidas no sean difíciles de entender o en el caso de las

tasas, complejas de poner en práctica. Si la fórmula matemática es complicada o varía mucho dependiendo del dinero que tenga el jugador, éste acabará por sentirse desmotivado y no querer jugar.

2.3.2. Dinámicas

Las dinámicas son el resultado o la puesta en acción de las mecánicas, es decir, el comportamiento de los jugadores, las decisiones que toman, el desarrollo del juego… que se deriva de la puesta en marcha de las mecánicas. No es algo fácil de concretar ya que no es tangible ni visible y sólo aparece mientras se desarrolla el juego (se conoce popularmente como *gameplay).*

Conceptos como el desafío o el compañerismo aparecen en este apartado. Por ejemplo, existen juegos de equipo en los que para ganar una partida contra otro equipo es recomendable o incluso a veces imprescindible jugar en equipo, provocando que los miembros del mismo equipo deban colaborar, cada uno dependiendo de su papel en el juego. Esto no quita que en los mismos juegos exista la posibilidad de jugar solo y ganar solo, ofreciendo al jugador diferentes modos de juego. Por otro lado existen mecánicas muy precisas previamente calculadas que ofrecen un reto al jugador y lo enganchan para que este intente superarlo una y otra vez, siendo consciente del proceso de aprendizaje que lleva y de que es un desafío factible.

Está parte del desarrollo del juego es muy importante, porque no importa qué tan precisas sean las mecánicas o qué tan variadas sean, si las dinámicas que derivan de las mecánicas son pobres, aburridas o provocan frustración en el jugador, es un fracaso. Un ejemplo muy popular sucede en algunos mapas de algunos *Shooters* multijugador. En estos juegos, existe un concepto, el *Spawn,* que significa literalmente engendrar o generar algo. Este concepto hace referencia a en qué parte del mapa se generan los jugadores al principio de la partida y cuando reaparecen después de morir. Para determinarlo, se calcula la posición del mapa teniendo en cuenta la situación de los enemigos, la de los aliados y también si es un modo de juego en el que hay bases o cualquier lugar que pertenezca a un equipo, se tiene en cuenta esta posición. Todo ello se combina para crear un punto de reaparición que cumpla que el jugador no muera al poco tiempo de empezar y que tampoco esté mucho tiempo hasta que localice el primer enemigo.

Pues bien, existen en ciertos juegos algunos mapas y/o modos donde el spawn está mal diseñado y, o bien el jugador no puede moverse más de 1 segundo cuando reaparece antes de morir (técnica conocida como *Spawn Trap*), o bien tiene que moverse durante muchos minutos antes de encontrar algo de acción. En ambos casos se genera frustración en el jugador, que no disfruta del juego y siente que pierde el tiempo.

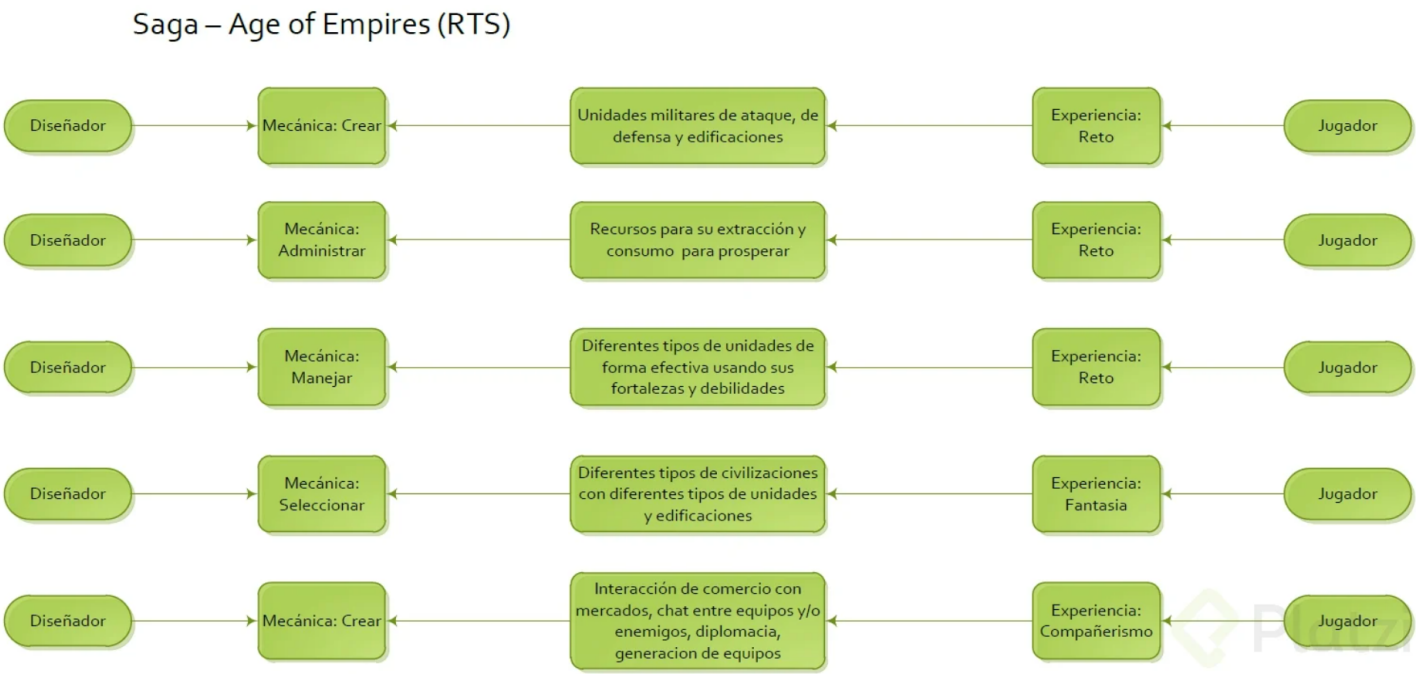


Figura 8: Ejemplo modelo MDA Age of Empires

2.3.3. Estéticas

La capa de la estética no se refiere al apartado artístico, a pesar de lo que a primera vista puede dar a entender. Esta capa hace referencia a la experiencia que tiene el jugador en el juego, al conjunto de sensaciones que provocan en él las interacciones que realiza con el juego.

Es importante desmarcarse de conceptos como “diversión” o “entretenimiento”. Estos conceptos son muy generales y tienen distintas interpretaciones según la persona a la que se pregunte. Además, todo juego se supone divertido.

Por ello los autores indican varias respuestas emocionales que se buscan a la hora de desarrollar un juego, como el desafío, el compañerismo, el placer, la tensión dramática de la narrativa…

Una muy buena imagen que ofrece el artículo *MDA: A Formal Approach to Game Design and Game Research* es la comparación de los juegos Charades, Quake, Los Sims y Final Fantasy. Con ello se expone que, aunque los 4 juegos sean considerados “divertidos”, lo son por muy diferentes razones. Charades es un juego de adivinar palabras haciendo mímica, Quake es un shooter en primera persona, Los Sims es un videojuego de simulación de una persona en una casa y Final Fantasy es un RPG (videojuego de rol) en el que el jugador maneja un personaje en un mundo de fantasía.

Mientras que Charades enfatiza en la expresión y el compañerismo, Quake provee al jugador de una sensación de competición y de desafío. Por otro lado, Los Sims deja al jugador la posibilidad de explorar y descubrir todas las posibilidades que puede llevar a cabo a la vez que éste decide el destino de su personaje. En último lugar, Final Fantasy ofrece un mundo de fantasía con una historia que pretende sumergir al jugador y además le proporciona desafíos que debe superar para avanzar.

Por todo esto, es sustancial definir y tener claro a priori qué sensaciones se quiere provocar en el jugador antes de comenzar a desarrollar un juego.

Para terminar, es importante señalar que el framework MDA es un exoesqueleto para el diseño de videojuegos. No es una fórmula mágica que garantiza el éxito en cualquier proceso de desarrollo.

Simplemente permite al diseñador definir objetivos claros y entender el resultado de los diseños en el videojuego o de incluso anticiparse a ellos. Además, ayuda a entender el impacto que tiene una mecánica del juego en la experiencia del jugador.

2.4. Interacción juego/jugador

Hasta ahora se ha analizado el concepto de juego y el desarrollo del mismo. Sin embargo, uno de los elementos más importantes, sino el que más, es el jugador. Al final el jugador es el centro de la cuestión. Él es el objetivo final que el juego debe satisfacer y para ello debe cumplir sus exigencias. Para que el jugador se divierta y disfrute del juego, dicho juego debe ser capaz de generar alguna experiencia recreativa en el jugador. Obviamente no existe algo en concreto que deba tener un juego para entretener al jugador, ya que no a todas las personas les atrae lo mismo.

En el apartado Definición juego se exponía qué era lo que caracterizaba a un juego para ser llamado como tal. Ahora se va a explicar a través de qué elementos el juego provoca que el jugador disfrute y qué espera el jugador que tenga el juego.

Un elemento importante que el jugador quiere que tenga el juego es la concretitud y la consistencia de las acciones que puede realizar. Dicho de otro modo, que tenga en todo momento claro qué puede realizar y el resultado de lo que haga y que, además, el resultado de esa acción sea siempre el mismo, o que si cambia, ser plenamente consciente de dicho cambio.

También le es agradable tener control sobre el progreso del juego, explorar caminos u opciones alternativas, etc. No obstante, es necesario que tenga presente el objetivo principal que debe cumplir en el juego. Debe suponer un reto alcanzable y se debe proveer al jugador de los medios necesarios para lograrlo. Es importante mantener el equilibrio entre estos dos apartados, para evitar que el jugador se pierda en el propio juego y para evitar que se frustre ante la falta de control en el juego.

Es normal que los objetivos principales del juego sean a priori inalcanzables en etapas tempranas. Por ello es lógico pensar que el jugador busca cumplir objetivos menores poco a poco que le lleven a evolucionar y a conseguir los medios necesarios para cumplir objetivos más grandes. Asimismo, y volviendo a la parte de conocer qué está sucediendo en el juego, el jugador necesita recibir retroalimentación de las acciones que realiza y del progreso que va generando.

Por otro lado, el jugador busca desafíos y retos constantes. No hay nada peor que sentir que algo es monótono y repetitivo cuando, en teoría, debería ser divertido. Por ello es necesario lanzar desafíos al jugador que generen en él la necesidad de superarlos, cumpliendo por supuesto el principio de no generar un reto que acabe frustrando al jugador. [11]

El equilibrio en la dificultad del juego que evita que el jugador caiga en el aburrimiento o en la frustración se corresponde con el “canal de flujo” del psicólogo Mihály Csikszentmihalyi. Para él, este canal es un estado sensorial que provoca la absorción completa por la actividad que está realizando la persona debido al placer y al disfrute que procesa. Esto deriva en que el tiempo pasa sin percatarse y nuestro cerebro procesa información de manera contínua sin pensar en otra cosa. Esta actividad puede ser contínua o puede tener pequeños altibajos en los que ciertas partes provoquen más tensión (como podría el enfrentamiento con un jefe especialmente fuerte) y otras sean más relajadas y asequibles.

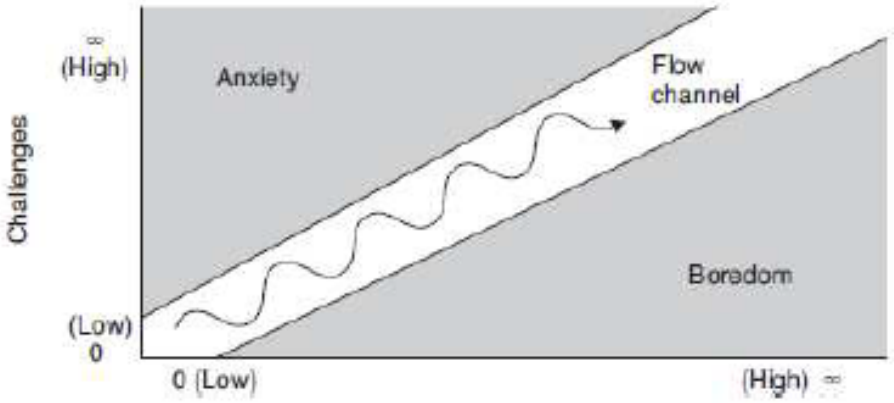


Figura 9: Alternativa teoría del flujo por Jesse Schell

Saliendo un poco del apartado personal, es evidente que hay diferencias de gustos entre distintos segmentos de la población. Los intereses cambian según se crece en edad e incluso existen ciertas preferencias dependiendo del sexo de la persona.

Existe un estudio de la mano del escritor inglés Richard Bartle, conocido por escribir numerosos escritos sobre el diseño de videojuegos en línea y multijugador, que aportaba una división en 4 tipos de jugadores (Figura 10) . Por un lado existen los triunfadores, que son simplemente los que buscan con ansía superar los objetivos principales del juego. También existen los exploradores, aquellos que lo que buscan es ver e interactuar con cada rincón del juego. Por otro lado existen los jugadores *sociales*, que lo que buscan es relacionarse con otros jugadores en el ambiente que caracteriza al juego. Por último se encuentran los asesinos, que son los jugadores competitivos por excelencia y lo que buscan es ganar a otros jugadores.

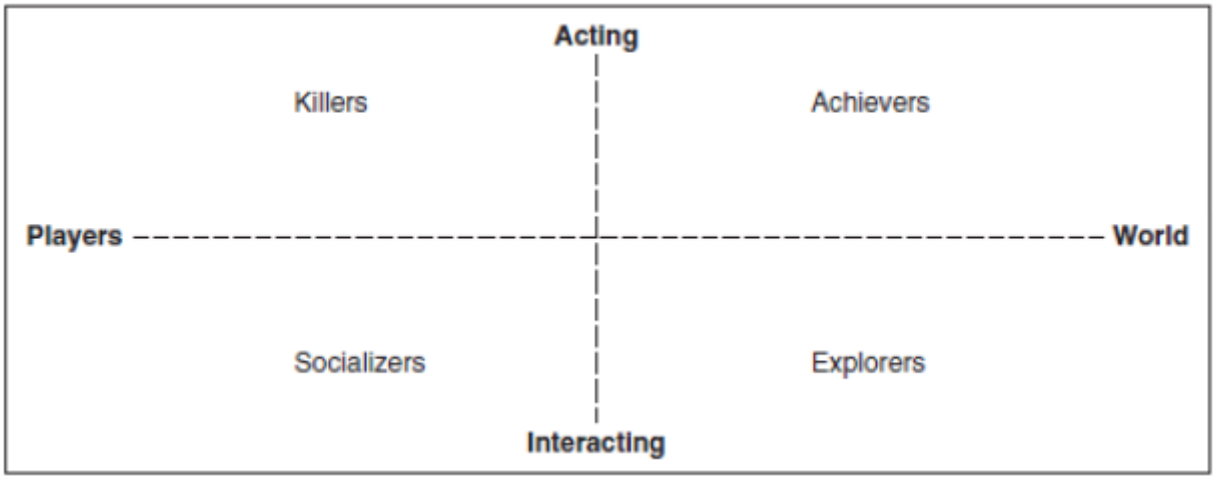


Figura 10: Tipos de jugadores según Richard Bartle

Hay más estudios que proponen nuevas divisiones o subdivisiones de las anteriores, pero la tónica general que se promulga es la del párrafo anterior. Lo importante a entender es que no se puede crear un juego que satisfaga por completo a todo tipo de jugadores. Por ello es importante recalcar de nuevo que hay que planear con detalle a quién va dirigido el juego, qué personalidad tienen y, con ello, definir las mecánicas que deriven en las dinámicas y las estéticas.

Por último, un apartado muy importante de la interacción entre el jugador y el juego es la interfaz. Es de vital importancia que el juego tenga una buena interfaz para que la experiencia del jugador sea buena. Es lógico pensar que en ella deben aparecer los objetivos o las acciones que el jugador puede realizar y que estás se presenten de acuerdo a su rango de relevancia según que momento. Sin embargo, es importante no saturar la interfaz con numerosos elementos ya que puede llevar a la confusión del usuario. Por ello es importante buscar el equilibrio entre mostrar toda la información relevante y mantener la interfaz lo más simple posible. [12]

2.5. Elementos del juego

A continuación se van a nombrar y a desarrollar los distintos elementos básicos que existen en un juego de la mano del desarrollador de videojuegos Jesse Schell. Existen otras clasificaciones de diferentes autores pero esta es considerada una de las más sencillas y completas.

2.5.1. Historia y Personajes

La historia es un elemento no interactivo y lineal, el espectador no interviene en ella y su único papel es el de observador. Esto es algo característico en películas o series.

No obstante, es muy habitual encontrar historias en videojuegos. Esto se debe a que proporciona una interconexión entre los demás elementos del juego, refuerza la temática y, en muchas ocasiones, incluso suele ser la protagonista del juego y los demás elementos giran en torno a ella. Incluso en juegos en los que no hay historia, es habitual que se escriban las historias de los personajes que hay en él, para justificar su apariencia y personalidad y, además, para que el jugador simpatice con ellos.

Y es que los personajes pueden llegar a ser el centro de todo el juego. Al igual que en películas y series, estos representan ideas o actitudes que los jugadores desean tener o comportamientos que idealizan. Hace años los personajes representaban ideas más simples como ser el héroe o ser el malo, pero hoy en día existen personajes con una gran proyección psicológica detrás que pueden llegar a cautivar al jugador, más aún cuando pueden controlarlo en el videojuego.

Por otro lado, en muchos juegos se deja al jugador la posibilidad de crear desde cero el aspecto físico de su personaje. Esto ayuda a que el jugador se sumerja en el videojuego desde el principio, ya sea creando un personaje similar a él, o creando uno el cuál quiere ser dentro del videojuego.

2.5.2. Estética

La estética abarca todo el apartado artístico, visual y sonoro del juego. Si bien es cierto que no importa cómo de espectacular se visualice un juego o qué tan maravillosa sea la música o los efectos de sonido si a fin de cuentas el juego no contiene las mecánicas apropiadas para ser disfrutado, no deja de ser uno de los apartados más cruciales a la hora de determinar si un juego es bueno o no.

Es más, existen determinados casos en juegos y videojuegos los cuáles son realmente aclamados como obras maestras por tener un apartado visual y sonoro espectacular y otros juegos que han sido duramente criticados por lo mismo.

Un ejemplo cercano es el videojuego The Legend of Zelda: Breath of the Wild el cuál fue muy bien valorado a su salida por tener un apartado artístico muy bien trabajado, que combinado con la historia y la jugabilidad, hacían de él un juego muy disfrutable. Por otro lado el videojuego Pokémon Legends: Arceus, que salió a la venta meses después y para la misma consola fue criticado por muchos de los jugadores al tener elementos visuales muy pobres en comparación con BOTW y no tener prácticamente elementos sonoros.

2.5.3. Mecánicas

Ya se han definido anteriormente. Comprenden las reglas del juego y definen la jugabilidad del mismo. Dictan las acciones que puede realizar el jugador y sus posibles respuestas a los eventos del juego.

Uno de los aspectos fundamentales cuando se definen las mecánicas en un juego es el propio balance de esas mecánicas para conseguir un equilibrio en la jugabilidad.

Este equilibrio que se debe alcanzar tiene varias ramificaciones en concreto:

Imparcialidad

Quizás una de las cosas que puede generar más indiferencia en la mente del jugador cuando se encuentra jugando es tener la sensación de que no importa qué haga, ganar o avanzar en el juego no depende de sus acciones, sino que está en gran parte determinado por el azar. Un juego cuyas mecánicas están en gran parte basadas en la aleatoriedad no invita al jugador a sumergirse en el mismo.

La manera más sencilla de garantizar un balanceo en la imparcialidad es la distribución de recursos entre los distintos jugadores por igual, garantizando que todos ellos empiezan el juego en igualdad de condiciones y que son puramente sus acciones a lo largo de la partida las que les acercan o alejan de la victoria. Sin embargo, en muchas ocasiones es interesante incluir cierta asimetría para generar pequeñas diferencias que estimulen el afán del jugador por conseguir sus objetivos.

Un ejemplo bastante claro de este último punto son los videojuegos Auto Chess, los cuáles han ido ganando popularidad en los últimos años. Estos juegos se desarrollan en un tablero similar al del ajedrez. La diferencia con respecto a este es que las piezas se van colocando poco a poco y que cada una tiene características muy concretas. Además existen numerosas combinaciones entre piezas, objetos, etc que generan bonús o ventajas para conseguir un equipo más fuerte y poder ganar la partida.

Figura 11: Partida en el videojuego Auto Chess TFT

La imparcialidad viene cuando llega la hora de comprar estas piezas o de adquirir estos objetos. Esto tiene un componente aleatorio ya que las opciones que se presentan en cada momento son inciertas, aunque con cierta probabilidad dependiendo del momento de la partida.

Dificultad

La dificultad es quizás el aspecto más difícil de balancear y de cuantificar en un juego. No obstante es también probablemente el más vital, ya que mantener al jugador en el Canal de Flujo es determinante para generar en el jugador el deseo de seguir jugando.

Alguno de los trucos para conseguir esto es incrementar la dificultad en cada nivel, ya que el jugador tendrá un mejor manejo adquirido con la experiencia, crear retos especiales que suban un peldaño extra en la dificultad pero no tanto como para generar frustración, dar cierta libertad al jugador para elegir qué hacer en cada momento o qué nivel realizar…

Toma de decisiones

La libertad del individuo para elegir qué hacer en cada momento del juego es otro aspecto claramente importante a la hora de balancear la jugabilidad. El número de elecciones que puede tomar el jugador no debe ser ni muy numeroso de modo que resulte abrumador ni muy pequeño, haciendo que su interacción con el juego sea mínima.

Por otro lado, las elecciones que tome el jugador deben tener un sentido en el desarrollo de la partida. No debe haber una elección o una serie de elecciones que siempre conduzcan a la victoria o que continuamente generen ventaja para el jugador con respecto a otras (por ejemplo, un arma es en cualquier situación, mejor que el resto).

Estrategia vs Mecánicas

Aunque existen videojuegos en los que predomina claramente uno de los dos aspectos, en la mayoría de ellos existe un balance entre las mecánicas del jugador (reflejos, rapidez de movimiento de las manos, etc…) y la estrategia de la partida. Este equilibrio es muy importante ya que permite que tanto jugadores que sean muy buenos en el apartado estratégico, como jugadores que sean mejor en el apartado puramente dinámico, tengan la posibilidad de conseguir la victoria, partiendo desde distintas partes.

Competición y Cooperación

Los juegos competitivos, en los que la victoria de uno o varios individuos suponen la derrota de otros tantos, generan una sensación de disfrute en los ganadores la cuál no surge en juegos plenamente cooperativos, en los que el jugador consigue la victoria junto a otros y nadie es derrotado.

La combinación de estas dos facetas en juegos competitivos en equipo es una buena manera de mantener esta sensación de placer que genera conseguir la victoria frente a otras personas y además añadir las dinámicas de trabajo en equipo y coordinación que supone la cooperación con otros jugadores para alcanzar el éxito.

Duración

La duración del juego no sólo viene determinada por la duración en sí misma de una partida, sino también por el tiempo que sucede desde que se comienza la partida hasta que el jugador o jugadores son plenamente conscientes de cuál va a ser el resultado final de la partida.

El jugador puede verse frustrado si pierde o sabe que va a perder en un momento muy próximo al inicio de la partida o muy aburrido si ocurre lo contrario, si sabe que va a ser el ganador mucho antes de que realmente suceda. Un ejemplo muy básico es el juego Monopoly, en el cuál llega un punto de la partida en el que las probabilidades de que el ganador sea un jugador en concreto son muy altas aunque quede mucho tiempo para que el juego termine.

Recompensas y Castigos

Son las formas más sencillas de indicar al jugador que lo está haciendo bien o mal. Ambas sirven para motivar al jugador a seguir jugando, tanto la satisfacción de recibir una recompensa sirviendo de prueba de que el jugador está desarrollando una buena partida, como la excitación de saber que correr un riesgo supone un castigo. Normalmente aparecen ambos juntos y están equilibrados de manera que a más desafiante es el reto, más satisfactoria es la recompensa y/o más severo es el castigo, aunque es importante no generar desmotivación con este último.

Algunas de las recompensas más usadas son el sistema de puntos o rango que muestre qué tan bueno es el jugador en ese juego, el desbloqueo de nuevas zonas, modos o utensilios, o de insignias, elementos estéticos, medallas… Por otro lado y en contraposición, los castigos más comunes son la pérdida de puntos, reducción de recursos como dinero o munición… También se puede privar de ciertos beneficios o de características conseguidas pero se recomienda que sea algo temporal, ya que puede ser muy frustrante perder algo que te ha costado mucho tiempo conseguir.

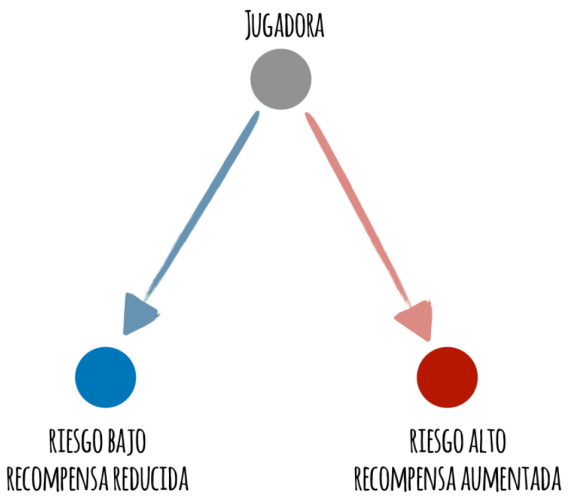


Figura 12: Balance riesgo/recompensa

2.5.4. Tecnología

El soporte sobre el que juego va a ser implementado es también un aspecto crucial en el desarrollo del juego. Obviamente no es lo mismo desarrollar un videojuego para realidad virtual que hacer un juego en el que los únicos elementos sean papel y lápiz y un pequeño tablero. Estos elementos determinan las posibilidades del desarrollo y condicionan las mecánicas y estéticas del juego. [13]

2.6. Creación de un videojuego

Tras haber definido todo lo que hay que saber de los videojuegos y de los juegos en general es hora de dar paso de forma muy breve a las diferentes etapas que puede tener este proceso y cómo deben ser abordadas desde el punto de vista del diseñador

2.6.1. IDEA

Ciertamente, no existe ningún guión escrito ni ninguna fórmula mágica cuyo uso derive en una idea revolucionaria y exitosa. Las ideas que pueden surgir son fruto de nuestra experiencia y del estudio que podamos realizar sobre el mundo que nos rodea. Es por ello por lo que normalmente la idea principal de un videojuego surge de haber jugado a otros o de encontrar algo en el entorno que pueda ser replicado en un videojuego, aunque con cuidado, ya que no hay que confundir la inspiración con copiar o repetir algo.

2.6.2. Desarrollo

Al igual que en todas las disciplinas en las que hay que desarrollar un producto complejo, la clave del desarrollo es el trabajo en equipo y que el equipo contenga personas con características multidisciplinares. Normalmente es mejor tener en el equipo a personas cada una especializada en su área, que tener un grupo con las mismas características, que tenga conocimientos sobre todo y que participe en todas las partes del desarrollo. Con esto no quiero decir que no se deba saber un mínimo sobre un campo que no sea personal (de hecho es necesario para poder aportar ideas), pero es cierto que se suele preferir un perfil especializado.

Ya se ha expuesto que el proceso de desarrollo es iterativo e incremental. El inicio consiste en pensar qué juego se quiere crear, sus objetivos y qué sensaciones o qué experiencia se quiere transmitir al jugador, que será el cliente. Es muy importante dejar claro este apartado para que en futuros pasos no haya dudas de qué se tiene que desarrollar.

Una vez definidas las estéticas del juego, hay que crear las mecánicas que deriven en las dinámicas que provoquen la experiencia deseada en el jugador.

Tras las mecánicas, se tiene un prototipo funcional, que es jugable. Hay que probarlo y analizar dichas pruebas para ajustar errores, bugs o regular la dificultad del juego. Este prototipo no tiene porque ser digital. En muchas ocasiones, el papel y boli son una herramienta muy potente para dar vida a las ideas y pueden ayudar a detectar problemas o inconvenientes de

manera muy rápida. Además, existen herramientas y programas que proporcionan los medios para hacer prototipos.

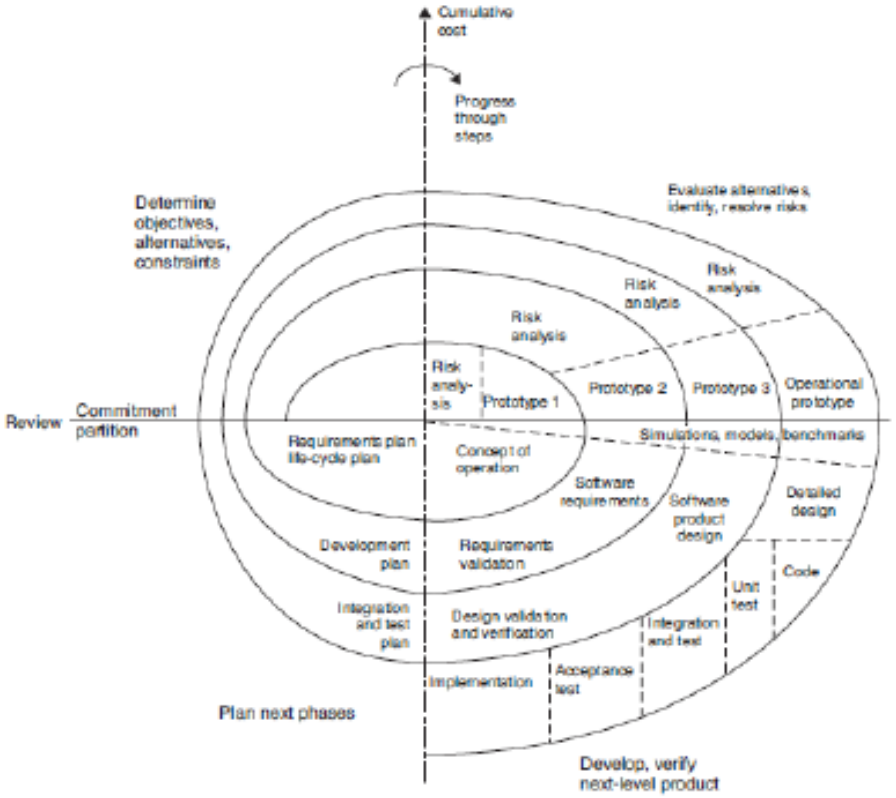


Figura 13: Proceso iterativo e incremental de desarrollo

Para terminar, se tiene que hacer una lista de los cambios a realizar tras analizar las pruebas y también de nuevas funcionalidades o requisitos que se quiere que cumpla el juego. Cuando se tenga, se vuelve al paso 1 y se comienza de nuevo.

Algunas buenas prácticas que se recomiendan seguir son las de tener siempre una versión jugable del producto y no esperar a la siguiente iteración para arreglar la versión anterior. Además, es de suma importancia documentar todo el proceso a medida que se sucede el desarrollo, y no dejar esta parte para el final.

También es más eficaz desarrollar el juego por módulos separados, para probar cada parte de manera aislada. Un ejemplo muy común en el desarrollo de videojuegos es separar el apartado artístico del gameplay. Normalmente, los desarrolladores realizan las pruebas con un personaje que es literalmente un cubo. Más tarde ese cubo será el personaje que los artistas están creando, pero para probar como funciona el gameplay, con un cubo es más que suficiente. Otro ejemplo en videojuegos de mundo abierto es el de desarrollar por zonas, de manera que se puedan probar por separado sin afectarse la una a la otra.

Es importante entender que este proceso tiene un fin y que es inconveniente refinar al máximo el videojuego. No es una buena práctica perseguir la perfección (ni en esta disciplina ni en cualquiera), porque esto llevaría a un proceso de desarrollo prácticamente infinito. Por suerte, existen por norma general plazos de entrega y presupuestos ilimitados, por lo que estos son los que la mayor parte de las veces marcan el final del desarrollo.

2.6.3. Pruebas

En el mundo del desarrollo software, los test no son algo opcional, deben ser realizados de la mano del propio desarrollo del producto, no solo para evitar que en el lanzamiento de este surjan errores o bugs, sino para evitar dolores de cabeza y malas prácticas. [14]

Es por ello que las pruebas son fundamentales, ya que nos permiten garantizar y demostrar que el código que se ha escrito se comporta de la manera que se había pensado y que cumple con los requisitos previamente definidos de funcionamiento y calidad.

Las pruebas se pueden catalogar según el contexto en el que se aplican en diferentes tipos.

Pruebas unitarias

Como su propio nombre indica, una prueba unitaria busca probar un componente o método aislado del resto del código y verificar que su comportamiento es el esperado. Estas pruebas se suelen realizar antes de integrar todos los componentes y suelen escribirse un poco antes de crear el propio componente, para tener muy presente a la hora de programarlo cómo debe responder ante determinadas entradas.

Pruebas de integración

Las pruebas de integración se realizan a la hora de juntar todos los componentes individuales para formar otros más grandes. Son muy útiles ya que el hecho de que las componentes de manera individual no generen ningún problema no quiere decir que a la hora de juntarse todo siga igual.

Pruebas de regresión

Estas pruebas se realizan cuando se modifica una componente de un proyecto, ya que con estas modificaciones surge la posibilidad de que el código existente no funcione como lo hacía anteriormente. Es importante destacar que no solo hay que probar las modificaciones realizadas, sino también el resto del programa, para garantizar que dichas modificaciones no perjudiquen al resto del código.

Pruebas de estrés

Las pruebas aisladas o con un número de usuarios o datos limitado puede no desvelar problemas que sí existen en el uso de la aplicación cuando está pasa al despliegue con usuarios reales. Por ello, es importante que, al igual que las pruebas unitarias se hace con análisis de valores límite, el sistema en su conjunto sea probado bajo condiciones extremas y con una cantidad de datos y/o usuarios que roce la capacidad máxima que pueda soportar. De este modo se puede vislumbrar cómo se comporta el sistema cuando está sobrecargado y cómo tras esta sobrecarga vuelve a su estado normal de funcionamiento.

Pruebas de seguridad

Quizás las pruebas que primero se vienen a la cabeza a la hora de probar un sistema. Obviamente son fundamentales ya que sirven para identificar brechas de seguridad y posibles flaquezas en el sistema. Dentro de este tipo de pruebas se suelen utilizar las pruebas de caja negra o de caja blanca.

2.6.4. Documentación

Al igual que en cualquier otro proceso de desarrollo, la documentación que recoge las características de dicho proceso es fundamental. En el caso de los videojuegos no solo sirve como punto de observación del trabajo realizado durante el desarrollo, si no que además sirve como punto de partida para elaborar el manual de instrucciones que debe estar presente junto al videojuego.

Entre los apartados que debería tener una buena documentación se encuentran obviamente el diseño en el que se basa el juego y toda la parte relacionada con la ingeniería detrás de las mecánicas programadas. Además debe incluir información sobre los objetivos del juego, las interfaces y todo lo relacionado con el apartado artístico. Por último es imprescindible que la planificación económica del desarrollo y la gestión del tiempo disponible aparezca de forma clara y concisa.

3. Tecnología 3D

3.1. ¿Qué es la Realidad Virtual?

Valiéndose de la definición que proporciona la RAE, la realidad virtual es la *representación de escenas o imágenes de objetos producidas por un sistema informático, que da la sensación de su existencia real.* [15] Es decir, cualquier representación visual producida por un ordenador y que tiene tal realismo que es capaz de engañar al cerebro humano y conseguir que piense que ese contenido forma parte de la realidad.

Si bien en esta definición no se puntualiza, es cierto que para lograr esta inmersión sensorial en la realidad que proyecta el ordenador, es necesario el uso de dispositivos extra como gafas de realidad virtual, cascos o incluso guantes o trajes que se puedan utilizar para interactuar con el “mundo” que se muestra virtualmente.

La primera creación de la humanidad que cae dentro de esta categoría es el apartado conocido como Blue Box (Figura 14) en 1929. Este artefacto fue utilizado por el cuerpo de aviación de los EEUU para entrenar a los pilotos durante la Primera Guerra Mundial. El Blue Box era capaz de seguir y emular las órdenes del piloto y estaba diseñado para reproducir condiciones adversas de vuelo como factores meteorológicos o situaciones extremas de campo de batalla. [16]

Figura 14: Blue Box

En 1963, el inventor Hugo Gernsbac desarrolló una televisión portable sujeta a la cabeza (muy similar a las gafas de realidad virtual actuales) que ofrecían una experiencia de inmersión virtual. Si bien es cierto que el dispositivo no se lanzó nunca al mercado y que no respondía a los movimientos de la cabeza a la hora de mostrar las imágenes, su diseño sirvió de impulso para el desarrollo de los dispositivos actuales.

Unos años más tarde, en 1968, se desarrolló otro prototipo más revolucionario que si soportaba el movimiento de la cabeza, ajustando las imágenes para que siempre estuvieran al frente de los ojos. Sin embargo el aparato estaba formado por un brazo mecánico sujeto al techo, por lo que era muy incómodo de manejar y de ahí recibe su nombre: la espada de Damocles (Figura 15) .

Figura 15: Espada de Damocles

Además de esto, la propia NASA presentó unas gafas de realidad virtual en 1986 al público (Figura 14). Este dispositivo incorporaba además sensores de reconocimiento de voz y también de unos guantes e incluso un traje entero especial.

En los últimos años del siglo pasado, Sega y Nintendo lanzaron sus propias consolas utilizando la tecnología de realidad virtual. Por un lado, Sega tuvo que parar el desarrollo debido a obstáculos en el desarrollo técnico del producto y por otro lado, Nintendo no consiguió comercializar el producto (que además era mediocre) y su desarrollo fue un absoluto fracaso.

Figura 16: Gafas VR de la nasa en 1986

El comienzo del auge de esta tecnología vino dado de la mano de Palmer Luckey, el cual construyó un prototipo de gafas de realidad virtual en 2012 llamado Oculus, que aunaba una visión espacial en 3D con una rotación completa de 360º. 2 años más tarde Facebook compró los derechos de este prototipo por 2 billones de dólares y es a partir de aquí cuando empresas tecnológicas como Samsung, Google, Microsoft… vuelcan su interés en esta tecnología. [17]

3.2. Gafas Oculus Quest 2

Las gafas para las que se ha desarrollado el proyecto son las *Oculus Quest 2* (Figura 17). [18]

Estas gafas han estado a la venta desde el 13 de octubre de 2020 y surgen como mejora de las anteriores Oculus Quest. Lo más destacable es el nuevo procesador que integran, el Snapdragon XR2, un procesador dedicado especialmente a equipos de realidad aumentada y virtual con tecnología 5G, que además ofrece una resolución de 3K por ojo a 90 frames por segundo y es capaz de soportar la información de hasta 7 cámaras.

Otro aspecto destacado es el incremento de la RAM de 4 a 6 GB y que el almacenamiento interno pase de 128 GB a 256 GB para la versión más costosa de las gafas, con una reducción del precio de 100€ con respecto a las anteriores [19]. Además, el kit al completo ofrece las gafas, 2 mandos, 2 pilas de tipo AA, un cable tipo C, un cargador y un adaptador para las gafas y pese a todo esto, el kit es menos pesado que en la versión anterior y los periféricos cuentan con elementos que garantizan una comodidad mayor.

Figura 17: Gafas Oculus Quest 2

Hablando del apartado de la seguridad, el dispositivo cuenta con un “sistema guardián” que se configura cuando se comienza a usar las gafas y que sirve para delimitar un área de juego alrededor del jugador y evita que el jugador se salga de ese área cuando están inmerso en la realidad virtual.

Por último, debido a la adquisición de Oculus por parte de Facebook, deriva en la necesidad de tener una cuenta de Facebook asociada a las gafas para poder hacer uso de ellas. El proceso de inicio de sesión y vinculación es realmente sencillo y las gafas, junto a la aplicación para iOS o Android que se debe descargar, proveen de todas las indicaciones necesarias para llevar a cabo esta vinculación. [20]

4. Juegos relacionados

4.1. Mejores Juegos de Realidad Virtual [21]

4.1.1. The Walking Dead: Saints & Sinners

Este juego está inspirado en la popular serie de televisión The Walking Dead emitida por primera vez en el año 2010 y aún en emisión, ofreciendo por tanto un ambiente post apocalíptico en un holocausto zombie en la ciudad de New Orleans.

En él, el jugador deberá sobrevivir a los no muertos y a otras personas, buscando comida y demás recursos para sobrevivir, acabando con los no muertos que intentarán devorarlo y aliándose o enfrentándose a los distintos personajes y facciones que se encontrará a lo largo del videojuego, además de recolectar objetos y tesoros que le permitirán conseguir mejores armas y demás mejoras y al final, conseguir ventaja para nuevos enfrentamientos.

Figura 18: The Walking Dead: Saints & Sinners

A pesar de ser una copia de la serie, el videojuego es considerado uno de los mejores juegos de realidad virtual creados hasta la fecha. No solo por la capacidad que tiene de sumergir al jugador en este fatídico ambiente post apocalíptico, sino porque el jugador tiene gran libertad para tomar decisiones y dichas decisiones tendrán un gran impacto en el desarrollo del juego y en las relaciones que se puedan establecer con los otros personajes que hay en él. [22]

4.1.2. The Room VR: A Dark Matter

El clásico y popular juego The Room, conocido por los ingeniosos puzzles que tiene para ofrecer, también tiene una versión en realidad virtual y con ello consigue una transformación increíble y se inunda de un misterio que atrapa al jugador y lo sumerge por completo.

El juego es una aventura en la que hay que resolver la desaparición de algunas personas. Como detective, el jugador deberá utilizar su ingenio para resolver múltiples y desafiantes puzzles y con ello, resolver el misterio de las desapariciones que acechan a lo largo de un museo, una iglesia… y demás salas con un trasfondo enigmático de por sí.

Figura 19: The Room VR: A Dark Matter

Encontrarse en primera persona en estas salas con la posibilidad de interactuar con todos los objetos alrededor y comprobar cómo cada acción desencadena en otros hechos, acompañado de una ambientación musical de misterio, logra un realismo casi perfecto y hace que el jugador se meta por completo en la piel del detective y active al máximo su ingenio y destreza para completar todos los retos.

4.1.3. Vader Immortal

Como no podía faltar, existe una versión en realidad virtual ambientada en el universo de Start Wars. En ella el jugador tendrá la posibilidad de manejar espadas lásers, utilizar “la fuerza” y por supuesto, combatir contra el mismísimo Darth Vader.

El juego se divide en 3 episodios durante los cuáles el jugador será expulsado del hiperespacio, tendrá que resolver un misterio ancestral, entrenar y perfeccionar sus habilidades de combate, conocer nuevos personajes y explorar el universo entero, hasta dominar la Fuerza y poder conquistar el fuerte de Darth Vader y derrotarlo en combate.

Figura 20: Vader Inmortal

La duración del videojuego es de aproximadamente 5 horas y si bien es cierto que la historia es envolvente y sobre todo nostálgica para los más mayores, la mecánica del juego no es demasiado compleja ni profunda, aunque sí más que suficiente para sentirse como un verdadero jedi. [23]

4.1.4. Crisis VRigade 2

Un shooter no podría faltar entre los mejores juegos de realidad virtual, y es que en esta ocasión, el jugador debe enfrentarse a tiros contra terroristas e ir avanzando de escenario hasta llegar completar el juego.

Se consigue una sensación de realismo prácticamente completa ya que el jugador es quién decide si cubrirse tras alguna cobertura, asomar el arma para disparar a ciegas, o salir por completo para participar en el fuego cruzado. Sumando a esto el Aim Controller, un accesorio para PS4 que representa un arma de fuego, se consigue una inmersión completa en un universo de guerra y acción sin precedentes.

Hay que destacar que no es un juego con las mejores texturas del género y que las animaciones de los personajes no son demasiado fluidas y realistas. Además su excesiva dificultad puede provocar gran frustración entre los jugadores menos experimentados, ya que el juego no ofrece más alternativas que la de seguir intentando una y otra vez superar los diferentes escenarios, aunque esto resulta compensado con el modo multijugador. [24]

Figura 21: Crisis VRigade 2

4.1.5. Larcenauts

Un “Hero Shooter” multijugador de 6 contra 6 que hereda características del popular juego Overwatch, con múltiples personajes y roles, animación a color dinámica y fluida y un ambiente competitivo traído a la realidad virtual. [25]

Figura 22: Lacenauts

Cada personaje tiene sus propios armas, habilidades y dispositivos arrojadizos, que junto a diferentes estadísticas y demás elementos permiten personalizar a los héroes en el juego consiguiendo que la estrategia sea algo fundamental para ganar las partidas y que ir dando tumbos y disparando a todas partes como un loco no sea nada eficaz.

4.1.6. Beat Saber

Este juego combina a la perfección el concepto de juego musical en el que hay que bailar al ritmo de la música con la habilidad de reacción de cualquier juego de lucha con espadas.

El objetivo en el juego es romper unos bloques de colores que irán llegando al jugador a medida que suene la música. Cada bloque tendrá que romperse con una de las dos espadas que se porten, de forma que el color del bloque y la espada coincidan. Además, importará la dirección en la que se rompen, según la flecha que tengan los bloques.

Si bien es cierto que el juego no ofrece más experiencia, y que el repertorio de canciones es más bien escaso, ofrece una experiencia única para el jugador. También hay que decir que el juego es de principios del año 2019 y puede que sea por esto o por su simpleza pero, es de los más vendidos hasta la fecha. [26]

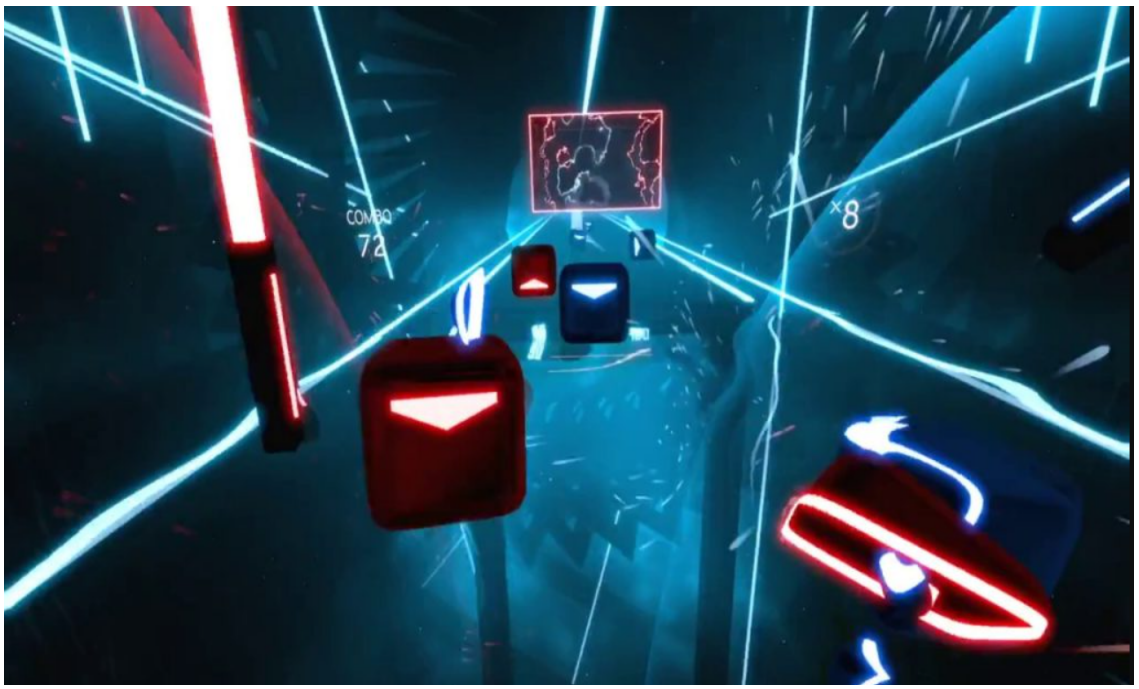


Figura 23: Beat Saber

4.1.7. Half-Life

Un FPS que narra la historia de una lucha titánica contra la invasión de una cruel raza alienígena, cargada con emoción e intriga y con unos gráficos y una inteligencia artificial y realismo de los enemigos soberbios. El juego ha sido aclamado por estas características y por la libertad a la hora de explorar el terreno y manipular las herramientas a tu alrededor. No en vano, este videojuego fue desarrollado de manera completa desde cero para realidad virtual.

Alabado por la crítica, el juego ha alcanzado mucha popularidad y sumerge al jugador en un universo único con una trama genial, haciéndolo uno de los mejores juegos de realidad virtual creados. [27]

Figura 24: Half-Life

4.2. MEJORES JUEGOS TOWER DEFENSE

El proyecto a desarrollar consiste en un videojuego dentro del género *Tower Defense* o Defensa de torres en Español. Este forma parte de los videojuegos relacionados con la estrategia y consiste en la defensa de una serie de territorios o enclaves u otra posesión de los ataques de enemigos. Es importante destacar que pese al nombre, esa propiedad que se debe defender no tiene por qué ser una torre, castillo o cualquier edificio que posea torres. Puede incluir personas, simples construcciones, etc.

La defensa se habilita normalmente construyendo edificios defensivos, que pueden ser torres desde las que se ataque con diversos tipos de armas. Sin embargo también se pueden habilitar barracones, cañones, hechizos… [28] [29]

4.2.1. ORCS MUST DIE! 3

En este juego el jugador controla un mago que tiene que defender fortalezas y castillos de la invasión de orcos acompañados de otra serie de criaturas infernales. Para ello, este mago podrá crear todo tipo de trampas, portar armas legendarias e invocar poderosos guerreros y hechizos, que deberá combinar utilizando diferentes estrategias.

Dentro del videojuego existe un modo historia con hasta 18 niveles y además misiones secundarias que suponen todo un reto para el jugador. Además existe un modo supervivencia en el que el jugador debe aguantar tantas oleadas como sea posible y lo que es mejor, existe la posibilidad de jugar en cooperativo con más jugadores.

Sin embargo lo que más destaca de este juego y lo que claramente lo diferencia de otros es el frenesí que experimenta el jugador debido al contínuo avance de los enemigos y a todas las posibilidades de trampas y construcciones que puede elegir para la defensa.

La combinación perfecta de disposición y organización del terreno entre oleadas, sumada al combate en tercera persona, hacen de este un juego muy frenético y estratégico y una experiencia sin igual para los jugadores que adoran los juegos “tower defense”. [30]

Figura 25: ORCS MUST DIE! 3

4.2.2. Dungeon Defenders

En este juego el jugador maneja un personaje y debe proteger un cristal de poder de oleadas de enemigos compuestas por orcos, demonios, dragones y demás criaturas fantásticas.

Similar al anterior, el jugador puede manejar su personaje para derrotar a hordas de enemigos o bien crear defensas con magia y estructuras. A medida que avancen las rondas podrá mejorar tanto su héroe como las defensas e ir desbloqueando nuevos hechizos y mejoras que le ayuden a superar los siguientes desafíos.

El juego tiene hasta 5 entregas, siendo la última de ellas del año 2020 y con la posibilidad de jugar en línea con hasta otros 4 jugadores para superar los retos más desafiantes. [31]

Figura 26: Dungeon Defenders

4.2.3. Plant VS Zombies

Este es sin duda uno de los más conocidos del género y un clásico a su vez. La idea es destruir oleadas de zombies que intentarán cruzar el mapa y dañar nuestra base.

Hasta ahí la idea puede parecer simple y similar a otros juegos, pero lo que hizo famoso a este en concreto y por lo que es tan conocido, es que las defensas de las que dispone el jugador son plantas, plantar de todos los tipos y colores que disparan, protegen o aturden al enemigo.

El juego provee de una acción exhaustiva y, literalmente, engancha al jugador a seguir completando los diferentes niveles e ir avanzando de escenarios. También dispone de varias entregas con diferentes matices en cada una de ellas y está disponible tanto en PC, como en móvil y en Playstation.

Figura 27: Plants VS Zombies

4.2.4. Kingdom Rush

Uno de los mejores juegos de la historia de los tower defense y un referente para muchos otros. En el clásico Kingdom Rush el jugador deberá proteger el reino de hordas de todo tipo de criaturas mágicas mediante la colocación en un terreno (previamente diseñado que los enemigos deben seguir) de diferentes tipos de estructuras como torres de magia, torres de arqueros, cañones y cuarteles de soldados, que atacarán a estos enemigos acabando con ellos y evitando que cruzan una línea de meta.

La saga cuenta con tres entregas: Kingdom Rush, Kingdom Rush Frontiers y Kingdom Rush Origins. A partir de la segunda se introduce un héroe que el jugador puede posicionar en el mapa y que le ayuda a superar las oleadas. Existen multitud de héroes que el jugador puede elegir y cada uno tiene habilidades diferentes.

A pesar de que no deja de ser un juego en el que tus unidades mueren una y otra vez, la paleta de colores viva y las animaciones hacen que el juego sea perfectamente aceptable para que los niños lo jueguen. Por otro lado, la dificultad varía mucho dependiendo del modo de juego seleccionado, con lo que resulta un gran reto para aquellos jugadores que elijan superar los modos más difíciles.

Figura 28: Kingdom Rush Frontiers

Este es sin duda un clásico del género y recoge toda la esencia de lo que debe transmitir un juego de tower defense, llevando al jugador al límite para pensar la mejor estrategia, basada en el momento y el tipo de colocación de las estructuras y también decidir qué mejora es la adecuada atendiendo al tipo de enemigos y el momento de la partida. [32]

4.2.5. Realm Defense

Otro videojuego muy popular sobre todo para dispositivos móviles y también con pinceladas épicas y heróicas como Kingdom Rush.

Este juego incorpora muchas más unidades fantásticas como dragones, magos e incluso dioses. Además ofrece hasta 300 niveles con dificultad incremental y que se desarrollan en bosques, montañas, praderas de hielo… y los recursos gráficos que consume no son muy elevados. Eso sí, gasta mucha batería. [33]

Figura 29: Realm Defense

4.2.6. Radiant Defense

Similar a los anteriores pero con la peculiaridad y la diferenciación de la ambientación futurista. Diferentes torres y armas compuestas en su mayor parte por todo tipo de rayos láser y explosiones de colores, que destruyen seres alienígenas que buscan invadir la tierra, y en este caso, empezar por la torre del jugador.

Una cosa que lo diferencia de otros juegos del mismo género es que cuando llega la hora de construir defensas, no se tiene un límite de tiempo hasta que aparece la siguiente oleada. Esto obviamente facilita superar los niveles, ya que tienes más tiempo para pensar en la estrategia ganadora. [34]

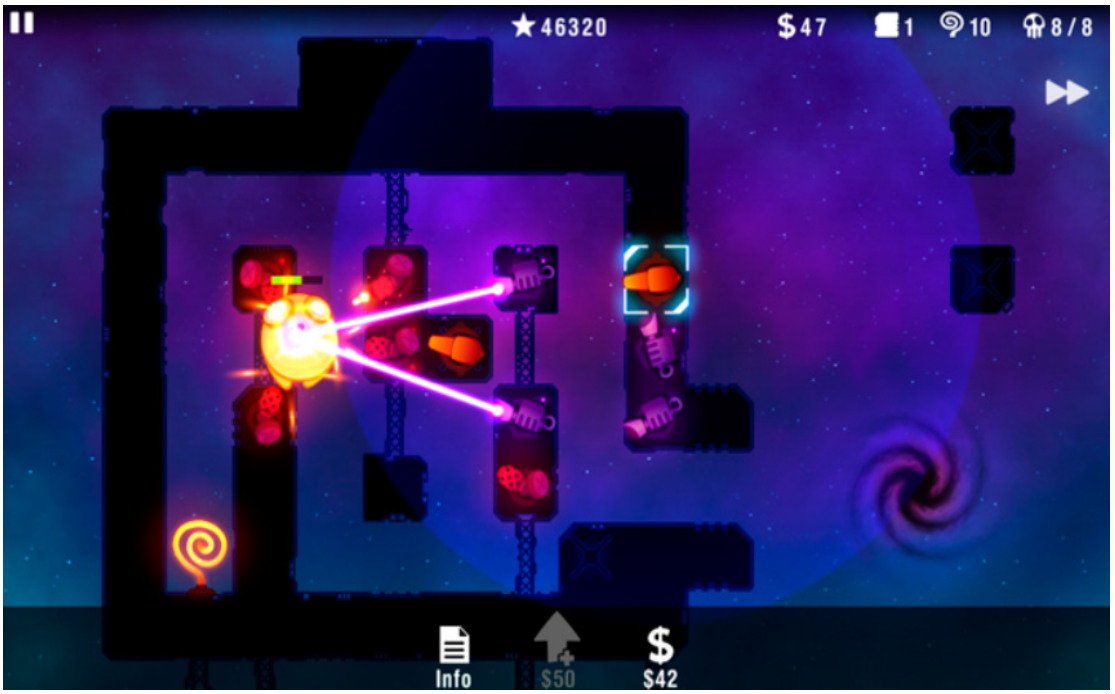


Figura 30: Radiant Defense

Lo más destacable del juego es el apartado artístico. El colorido y unos gráficos muy buenos hacen que este juego destaque en este género, además de la luz y las animaciones de los personajes dentro del juego, que están muy bien trabajadas.

5. ¿Qué es Unity?

Como ya se ha reflejado en el apartado anterior, Unity pertenece a la categoría de programas de “Motor de videojuego”. Esto debería dar a entender que Unity es un software específico para el desarrollo y la creación de videojuegos. Sin embargo, Unity también se ha utilizado para la creación de experiencias en Realidad virtual e interactivas e incluso series animadas cortas como “Baymax Dreams” [35] derivada de la película “Big Hero 6” producida por Disney junto a Unity y películas como “El Rey León” [36] (la de 2019).

Por tanto, quizás el nombre más correcto para esta herramienta sea **un motor de creación de contenido.** [37]

5.1. HISTORIA

Creado por Unity Technologies, su primer lanzamiento fue en la Conferencia Mundial de Desarrolladores de Apple en el año 2005 y estaba desarrollado para uso exclusivo en la plataforma Mac. Este nuevo software tenía como objetivo proporcionar un motor para el desarrollo de videojuegos a pequeñas y grandes empresas por igual, provocando un proceso de democratización del desarrollo del videojuego, de manera que tanto grandes compañías como desarrolladores, artistas o diseñadores individuales pudieran usar este software.

En un principio, había dos versiones del software: Indie y Profesional. La primera estaba destinada a aquellas pequeñas compañías o personas individuales que no dispusieran de un fondo económico relativamente grande y su precio era de 300 dólares. La versión Profesional costaba 1.500 dólares y traía con ella misma todas las funcionalidades del motor.

A medida que sucedieron los años, Unity pasó a ser compatible para iPhone, Android y, con el lanzamiento de la versión 4.0, se cerraron acuerdos con Sony, Microsoft y Nintendo para permitir la compatibilidad con todos sus sistemas, por lo que era ya posible desarrollar en plataformas Mac, Linux, iOs, Android, etc y por supuesto para todo tipo de consolas como PC, PlayStation, Xbox, Wii, WiiU…

Otro avance significativo que es indispensable destacar, es que en el año 2009, la compañía desarrolladora tomó la decisión de dejar la versión Indie de manera gratuita para todos los usuarios y que incluso con esta versión, se permitiese distribuir los juegos creados. [38]

5.2. CARACTERÍSTICAS

La herramienta de Unity contiene innumerables opciones que engloban el renderizado de gráficos en 3D y 2D, herramientas de networking para el desarrollo multijugador, herramientas de puesta en marcha de algoritmos PathFinding para Inteligencia Artificial, desarrollo multiplataforma (hasta 25 plataformas diferentes), procesos de exportación entre plataformas totalmente automáticos y, por supuesto, herramientas para la colaboración con otras personas trabajando en el mismo proyecto

Una de las mejoras cosas de Unity es la gran librería a la que tienen acceso los usuarios. En ella se pueden encontrar herramientas, tanto para 3D como para 2D, de todo tipo, como animaciones para incluir en personajes, multitud de criaturas con diferentes características, objetos que incluir en el entorno donde se desarrolle el escenario como mobiliario, vegetación, vehículos, etc y también elementos que ayuden a desarrollar una interfaz de usuario para todos los gustos y colores. También se tienen paquetes que presentan características adicionales relacionadas con el machine learning o servicios para controlar métodos de pago en el videojuego, ya sea en moneda regular o en criptomoneda.

Por otro lado, se tienen paquetes relacionados con el sonido ambiente, la música y los sonidos para efectos especiales y herramientas varias relacionadas con la inteligencia artificial, la cámara, la gestión del sonido y la animación, herramientas de diseño y modelado, sistemas de partículas y efectos varios, físicas, sprites, terrenos y paquetes que ayudan en la programación del videojuego y en el control de versiones.

Esta gran variedad realmente llama la atención y es cierto que un usuario se puede tirar horas e incluso días buscando exactamente qué paquete o asset necesita o cree que va a encajar mejor en el proyecto que está desarrollando. No obstante, lo mejor de todo es que existe una gran cantidad de contenido que es totalmente gratuito y no por ello de menor calidad que el que es de pago. Esto es debido a que la comunidad de desarrolladores de Unity aboga en su mayoría el software libre y existe una mentalidad general de compartir el trabajo realizado a cambio de posibles donaciones o de crédito para la o las personas que desarrollen el contenido de los paquetes.

No obstante, hay ocasiones en las que determinados paquetes de pago se encuentran en rebajas o que incluso son gratis por un pequeño periodo de tiempo, por lo que es recomendable visitar la asset store de Unity de vez en cuando, incluso cuando no se esté trabajando en ningún proyecto o no se necesite ningún paquete.

A continuación se van a describir diferentes elementos que proporciona el editor de Unity y que definen el funcionamiento de la herramienta y el proceso de desarrollo.

El primero es la Escena. Una Escena es simplemente un conjunto de objetos del juego que se combinan para crear un unidad sólida que puede ser un escenario de un nivel del juego, un menú concreto… Entre los objetos que la forman están elementos decorativos y de ambiente, obstáculos, efectos de sonido, partículas visuales y en definitiva, los elementos que forman el escenario de esa parte del juego.

Dicho esto, es importante definir que es un objeto en Unity. Un objeto en Unity (o GameObject, que es como lo llama la propia herramienta) son sencillamente contenedores de funcionalidad añadida. Por sí solos no logran nada, pero sirven para depositar otros componentes y crear objetos que tengan la funcionalidad y comportamiento deseado.

Otro concepto muy importante para el desarrollo es el de Prefab. Un Prefab es una plantilla de objetos que almacena las componentes de los mismos y permite la administración de las componentes de una manera más dinámica.

Es muy útil cuando se tiene la situación de que se quiere modificar una componente del objeto y dicho objeto está replicado varias veces en la escena. La modificación del prefab implica la modificación de todos los objetos que han sido instanciados a partir de dicho prefab, sin impedir por ello que un objeto en concreto pueda ser modificado si así se desea. En definitiva, el Prefab dictamina las características iniciales con las que se crea un objeto.

A medida que un proyecto se desarrolla y el número de objetos crece, es importante establecer una organización jerárquica agrupando unos objetos con otros según el interés del desarrollador. Unity proporciona dos tipos de herramientas para esto; los tags y las capas.

Una tag es una simple etiqueta asociada al objeto. Varios objetos pueden tener el mismo tag sin ningún problema. Las capas funcionan de manera muy similar, aunque existe un máximo de 32 capas disponibles. Con estas dos herramientas resulta muy sencillo a través de código acceder a los distintos objetos y manipularlos según se desee.

Unity proporciona mucha documentación para sus usuarios dividida en 2 manuales: uno general y otro específico para scripting [37].

5.3. INTERFAZ

La interfaz de Unity viene compuesta por todas las ventanas de la aplicación que ofrece el programa y que estén visibles en el momento de su uso. Estas ventanas pueden abrirse desde el menú de arriba en el apartado Windows. Se van a nombrar las ventanas que son más relevantes.

5.3.1. Hierarchy

La ventana de jerarquía muestra las escenas que se tienen abiertas en ese momento y todos los objetos que forman parte de dichas escenas. También brinda un menú sencillo que permite la creación de diferentes tipos de objetos, como efectos, video, audio…

5.3.2. Project

La ventana de proyecto muestra todas las carpetas y archivos que contiene el proyecto. Entre ellos se encuentran los scripts que guardan el código, todos los sprites, texturas y las propias escenas del proyecto. Además contiene los archivos o dependencias de terceros.

5.3.3. Console

Una ventana de consola que muestra mensajes con información relativa al proyecto y a la ejecución del mismo. Señala errores de compilación y se puede utilizar para imprimir mensajes y ayudar a depurar errores.

5.3.4. Scene

Esta ventana muestra la escena actual sobre la que se está trabajando. Carga todos los elementos de la propia escena en pantalla y permite la manipulación y la adición de objetos a la misma. Aquí se puede editar cualquier cosa del escenario en el que se va a desarrollar el juego.

5.3.5. Game

Esta ventana es la que muestra qué se ve cuando comienza a jugarse y depende de las cámaras que tenga la escena añadida.

5.3.6. Inspector

El inspector muestra las propiedades y características del objeto seleccionado. Permite además añadir y quitar componentes al objeto, los cuáles sirven para dotarlos de funcionalidad. Aquí es donde se pueden añadir los scripts que se han programado.

5.3.7. Project Settings

En esta ventana se puede administrar todas las configuraciones del proyecto, como el apartado sonoro, la calidad de los gráficos, el sistema de registro de acciones de entrada del usuario, la detección de elementos físicos en el propio juego o la gestión de plugins externos que han sido incluidos en el proyecto.

Existen muchas más ventanas con las que se puede trabajar con Unity, como la ventana de Audio Mixer con la que mezclar y editar los sonidos del proyecto o la de Navigation, que sirve para determinar zonas por las que los objetos seleccionados puedan desplazarse mediante inteligencia artificial. No obstante las definidas arriba son las básicas y fundamentales y con las que más hay que estar familiarizados a la hora del desarrollo

5.4. GRÁFICOS

El número de herramientas y de posibilidades que ofrece Unity para gestionar el apartado gráfico en un proyecto es gigantesco y realmente difícil de resumir en pocas páginas. De hecho existen personas que se dedican única y exclusivamente a este apartado en el desarrollo de un videojuego, ya que el conocimiento que hay que tener para sacar el máximo provecho a estos mecanismos de creación de contenido visual es muy grande y requiere de mucha especialización.

El primer elemento gráfico que se puede venir a la cabeza es la luz. Es incorrecto suponer que la luz está simplemente ahí, encima de los objetos del videojuego y que esta iluminación es la adecuada en todo momento. Unity ofrece distintos tipos de “objetos luz” que se adaptan a las necesidades del desarrollador en cada momento.

La luz más general es la luz direccional, que simplemente provoca que el escenario esté iluminado por igual, como si estuviese siendo cubierto por el sol. Es una luz a la que no le afecta la posición en la escena. Solo toma en cuenta la rotación, que hace que estos rayos de luz incidan desde un ángulo u otro sobre los objetos de la escena.

Si se desea utilizar una luz cuya iluminación depende de la posición del objeto entonces se debe usar el *Point Light*, que representa una bombilla y cuya posición e intensidad en el escenario afecta a cómo de iluminados están los objetos.

Por otro lado existen focos que, al contrario que las bombillas, no emiten luz por igual en todas las direcciones, si no que centran esta luz en un área en concreto en forma de cono, cuyo diámetro o alcance puede ser manipulado.

Se pueden encontrar más tipos de luz que sirven para determinadas ocasiones en concreto, pero estos tres mencionados arriba son los más importantes y con los que se pueden solventar la mayoría de problemas de iluminación.

Otro de los elementos fundamentales en el apartado gráfico es lo que se conoce como textura. Las texturas hacen referencia a los colores que se implantan sobre los objetos o sobre el escenario en general y consisten en bitmaps o mapas de bits en español. Estos mapas de bits contienen información binaria que sirve de representación de la paleta cromática general. En un bit se puede representar el color negro y blanco, pero la combinación de múltiples bit en un pixel de pantalla sirve para mostrar múltiples colores. Obviamente a mayor número de bits, mayor diversidad de colores se podrán representar.

Los materiales son otro de los elementos importantes en el desarrollo de videojuegos en general. Principalmente definen cómo el objeto que contiene el material debe ser mostrado. Permite la edición de características como la rugosidad, el brillo, la opacidad, etc. Las opciones disponibles para modificar el material vienen dadas por el shader que este contenga. Un shader es un script que contiene un conjunto de cálculos y algoritmos que determinan el color de los píxeles.

A la hora de crear efectos visuales como disparos, explosiones o cualquier elemento que forme parte de la ambientación lo más usado son los sistemas de partículas. Estos sistemas de partículas son en sí mismos efectos visuales con una serie de parámetros que se pueden editar para que esos efectos sean similares a los que el desarrollador desea. Se pueden modificar características como la duración, la rapidez de movimiento de las partículas, la repetición, el espacio en el que se mueven y además, se pueden utilizar materiales y shaders para personalizar estos efectos.

Otro de los elementos visuales importantes es la cámara. Evidentemente, Unity ofrece la posibilidad de introducir diferentes cámaras en una escena y es muy sencillo saltar de una a otra cuando se desee. Además ofrece herramientas de automatización del movimiento de la cámara y se puede pre programar directamente en el editor el recorrido de la misma y su velocidad y rotación.

5.5. FÍSICA

La física es quizás una de las componentes más importantes de un objeto en Unity y la que determina cómo realmente se comporta ese objeto. En definitiva, aproxima el comportamiento del objeto al que tendría en la vida real.

El motor de físicas que incluye Unity permite asignar magnitudes como la velocidad o la aceleración a un objeto, definir si está afectado por la gravedad y cuánta masa tiene o determinar qué sucede cuando es golpeado por otro objeto. Existen dos componentes que habilitan todas estas posibilidades con diferentes parámetros.

5.5.1. RigidBody

Este es el principal componente que controla el comportamiento físico de un objeto. Con ella se puede designar masa, aceleración y aceleración angular del objeto, además de determinar si está afectado por la gravedad y restringir su movimiento en los diferentes planos. A mayores ofrece la posibilidad de escoger en qué momento el objeto concreto está afectado o no por los parámetros físicos designados, permitiendo que el objeto pueda moverse por código sin desencadenar por ello conflictos con el apartado de las físicas.

5.5.2. Collider

Los colliders definen la forma que tienen un objeto para las físicas, es decir, el impacto que dicho objeto pueda sufrir no se produce cuando el objeto que impacta entra en contacto con su maya, si no que se produce cuando choca con este collider. Además se puede añadir physics materials (o materiales de física) que, de forma similar a los materiales, permiten añadir detalles al collider como la fricción o el rebote.

Por otro lado, existe una opción para detectar cuando un objeto entra dentro del espacio del colliders de otro objeto sin que ello desemboque en una colisión. Un objeto con una componente collider que esté configurada como trigger no se comporta como un objeto sólido y atraviesa los objetos con colliders con los que entra en contacto. Esto es muy útil sobre todo para saber cuándo un objeto choca con algo pero se quiere evitar que lo desplace o provoque una reacción en dicho objeto.

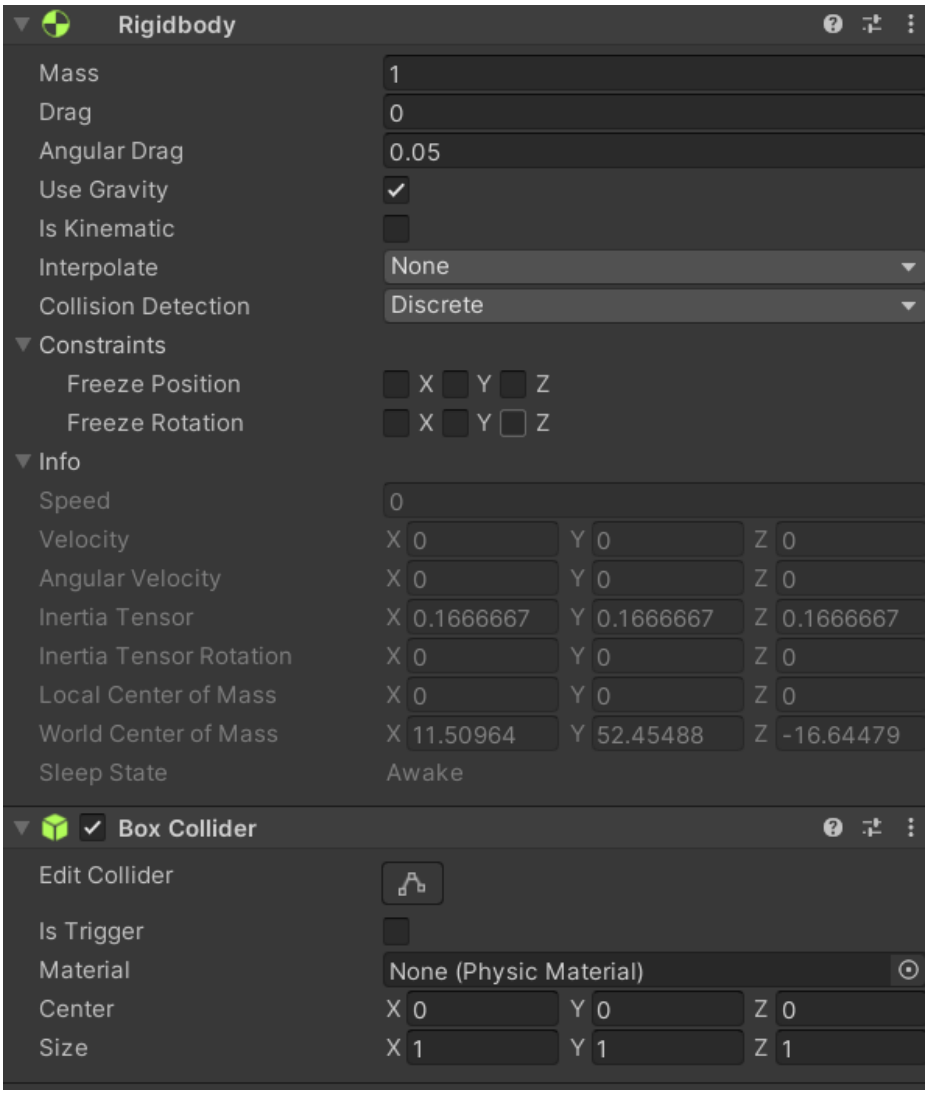


Figura 31: Componentes Rigidbody y Collider

Aunque estas dos componentes son las más destacadas existen muchas otras. Los Joints (articulaciones) por ejemplo sirven para pegar RigidBodies entre sí y según el tipo de articulación usada está unión tiene unas u otras características.

5.6. SCRIPTING

Los scripts de código son también algo fundamental a la hora de desarrollar videojuegos con Unity. Con estos se puede controlar las componentes que tienen los objetos en tiempo de ejecución, modificando sus valores cuando se quiera, e incluso crear nuevos componentes que agreguen características especiales a los objetos. Además permiten registrar los eventos de entrada por parte del usuario y desencadenar eventos en el juego.

Una de las primeras cosas que se debe aprender de la programación cuando se usa Unity es las posibilidades que te ofrecen las Funciones de Evento (Figura 32). Estas funciones son nativas de Unity y se ejecutan en un orden concreto y con un periodicidad determinada siempre y cuando estén declaradas en el script que se está utilizando.

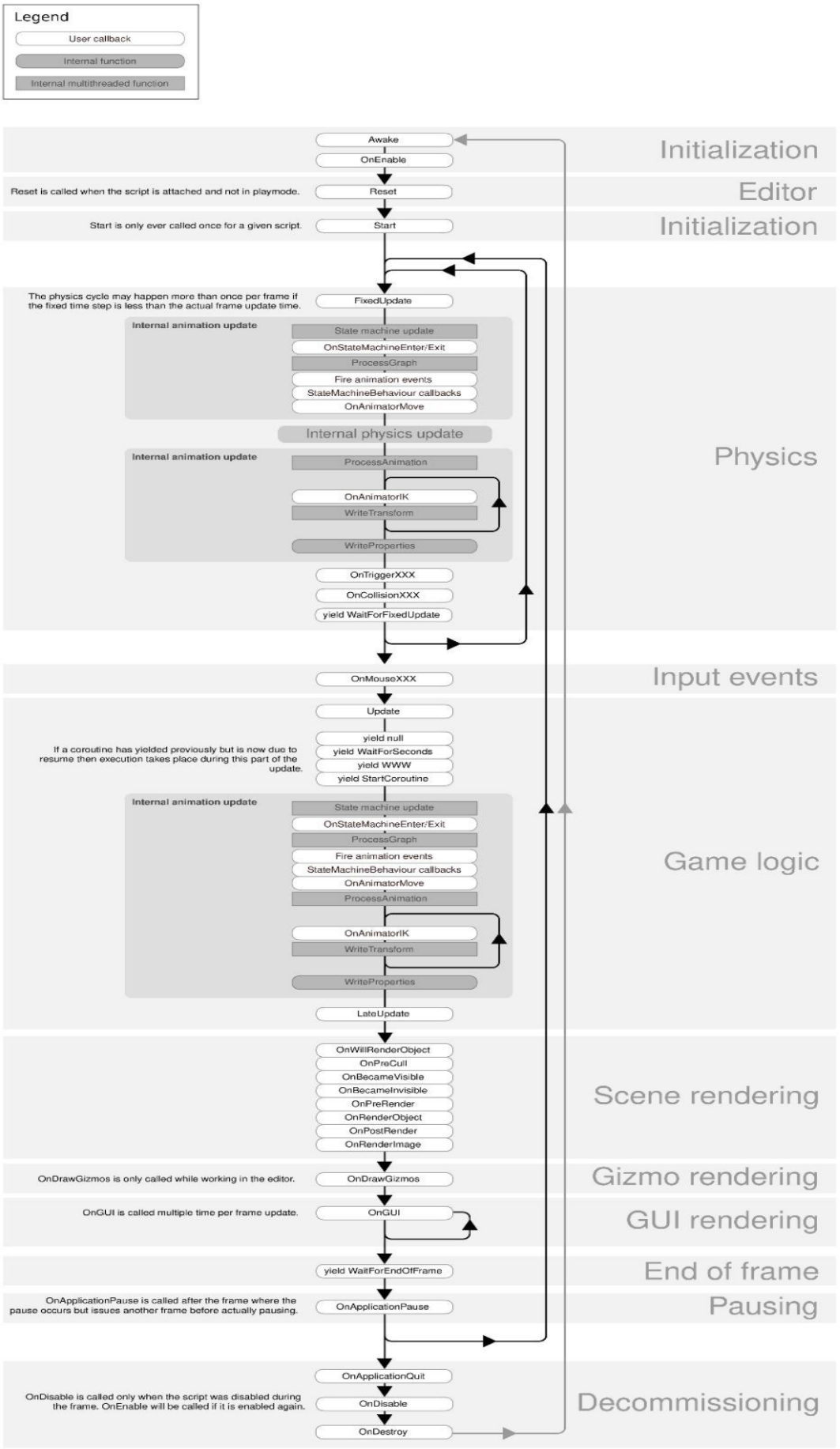


Figura 32: Diagrama de funciones nativas de Unity

Como se puede apreciar en la figura de arriba, existen multitud de estas funciones. A continuación se realizará una breve descripción de las más fundamentales para el correcto desarrollo en Unity.

5.6.1. Awake

El método Awake es el primero que se ejecuta. Solo se hace una vez y se llama cuando un objeto que contiene un script con dicho método en él se carga por primera vez en la escena, ya sea porque forma parte de ella cuando se carga o porque se instancia en tiempo de ejecución en ella con la orden Object.Instantiate.

Como este método se ejecuta justo después de que la escena se carga, normalmente se utiliza para detectar objetos en la escena, como por ejemplo saber cuántos tipos de objeto pertenecen a una capa u otra o para saber qué objetos están clasificados con un tag específico, y a partir de ahí, trabajar con esta información.

5.6.2. Start

Al igual que Awake, Start solo se llama una vez. Sucede justo después de la llamada a Awake y es necesario que el script esté activo en el objeto que lo contiene, no como con Awake, que se ejecuta sin importar lo que pase en el momento de la inicialización.

Esta función se utiliza para recoger información de otros objetos al estar estos inicializados. Si esto se hiciese en el Awake podría haber errores, ya que no se puede determinar en qué orden se `despiertan" los objetos en la escena y podría darse la casualidad de intentar acceder del objeto A a datos del objeto B sin estar este último aún inicializado. Como el método Start se ejecuta siempre detrás del método Awake, esto asegura de forma íntegra que los objetos están cargados en escena a la hora de acceder a ellos.

5.6.3. Update

El método Update es un método que se llama una vez por cada frame
de ejecución. Se usa típicamente para detectar cuando una variable toma un determinado valor y se quiere desencadenar un evento.
Existe una función paralela a esta llamada FixedUpdate que tiene la peculiaridad de que se ejecuta el mismo número de veces por segundo (según el valor de Time.fixedDeltaTime) sin importar a cuántos frames por segundo se está ejecutando la aplicación. Se usa sobre todo para cálculos que tengan que ver con la física incluida en el juego.

Es importante destacar que un mal uso de estos métodos puede acabar en cuellos de botella en la memoria del ordenador. Al ser funciones que se ejecutan periódicamente durante un intervalo de tiempo muy pequeño, la inicialización de variables o las operaciones que suponen almacenamiento de datos son dos claros ejemplos de malas prácticas en el desarrollo en Unity

5.6.4. OnDestroy

El método OnDestroy destruye el objeto cuando es llamado y si este objeto estaba activo en una escena. Esto quiere decir que lo elimina de la escena en la que se encuentra. Este método se llama por defecto cuando la escena en la que se encuentra es destruida o cuando se para el modo play en el editor de Unity.

Aunque normalmente se utiliza este método para, justo antes de destruir el objeto (porque es lo que corresponde con el flujo de la aplicación y ayuda a ganar espacio), guardar datos relativos

al mismo, desde la documentación de Unity se recomienda no depender del mismo. Esto es porque por ejemplo en dispositivos móviles, cuando el usuario suspende la aplicación, el sistema operativo puede cerrar la aplicación para liberar recursos y Unity puede no ser capaz de llamar a este método final antes de que esto suceda, llevando esto a una posible pérdida de datos. Es por ello que existen funciones para detectar esta suspensión de la aplicación, como OnApplicationFocus, que como su propio nombre indica, registra si el dispositivo móvil está o no usando una aplicación.

Estas cuatro funciones son los pilares del periodo de vida de un objeto en Unity. Si bien es cierto que ni mucho menos es necesario que aparezcan en todos los scripts que estén unidos como componentes al objeto (es más, es considerada una mala práctica tener estas funciones en un script cuando no realizan ninguna función útil), son en la mayoría de ocasiones de gran ayuda para su control durante el funcionamiento del juego.

5.7. ANIMACIONES

Otra característica fundamental para el desarrollo, no solo en Unity, sino en cualquier otra plataforma, son las animaciones. Estas definen el movimiento, o más bien, el comportamiento de los objetos en tiempo de ejecución.

En Unity existe un componente Animator (Figura 33) que es necesaria para que un objeto se comporte de acuerdo a unas animaciones asignadas. En este componente se puede asignar qué controlador de animaciones se va a usar y qué avatar se va a usar en caso de estar animando un objeto humanoide.

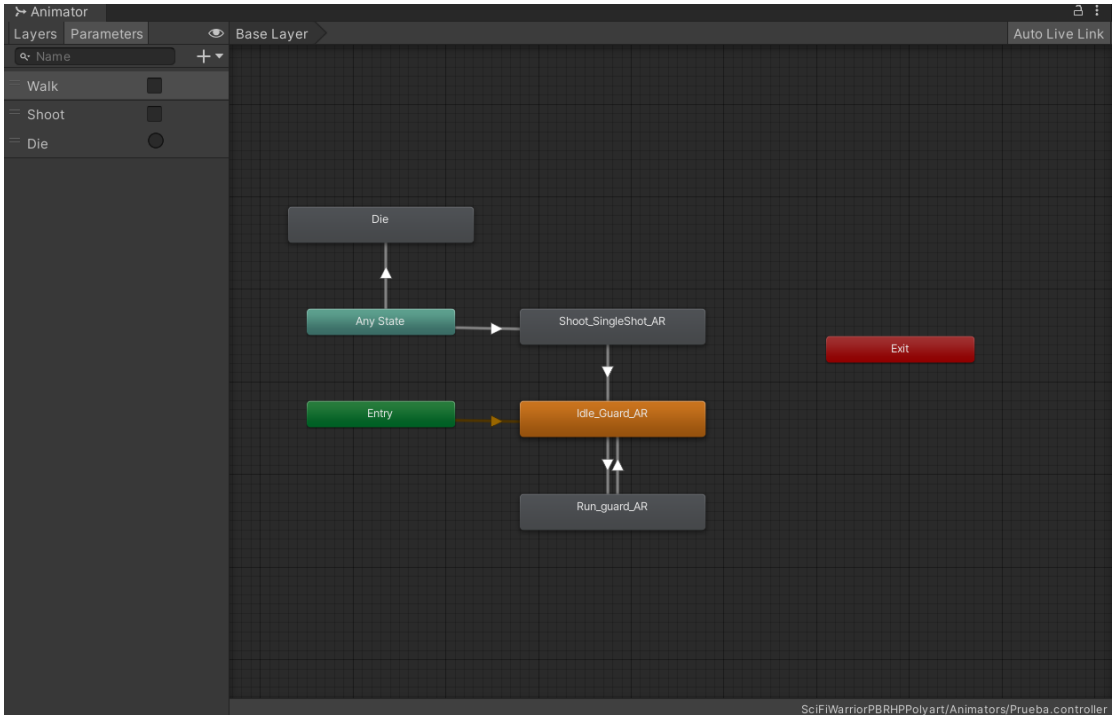


Figura 33: Ventana Animator en Unity

El controlador de animaciones es una ventana de Unity que permite agrupar varios clips de animaciones y marcar las transiciones de unos a otros de acuerdo a unos parámetros. Es básicamente una máquina de estados que incorpora tres estados por defecto; uno de entrada, que es el primero que se ejecuta y que supone el comienzo del conjunto de animaciones, uno de salida, que simboliza el final, la última animación que se realiza, y uno etiquetado como “cualquier estado”. Este último sirve para hacer transiciones hacia un estado concreto desde un estado cualquiera. Por ejemplo: cuando la vida

de un personaje llega a 0, no debe importar si dicho personaje estaba caminando o si estaba disparando a otro objetivo, siempre que su vida llega a 0 debería lanzarse una animación que de a entender que dicho personaje no tiene vida y que va a ser destruido. En este caso se puede usar el estado “cualquier estado” para realizar una transición al estado muerte y de esa forma lanzar esta animación sin importar qué animación se estaba llevando a cabo anteriormente.

Las transiciones entre estados se realizan normalmente mediante parámetros de tipo boolean o de tipo float, para detectar si una situación se cumple o no o para saber si una cantidad sobrepasa o no un umbral.

Cuando se quiere mezclar animaciones que tienen movimientos o gestos muy similares entre sí se utiliza una herramienta muy potente que incluye Unity: los Blend Trees. Estos permiten el acoplamiento de las animaciones de una forma suave utilizando los parámetros deseados, que marcan la “intensidad” de cada animación.

Quizás el ejemplo más conocido para usar esta herramienta es cuando se quiere controlar el movimiento de un personaje de acuerdo a su velocidad o a su dirección. La transición entre la animación de caminar y de correr, o entre caminar, trotar y correr más deprisa debería ser una transición fluida y sin cortes repentinos. De igual modo sucede con la dirección del personaje. Normalmente los clips de animación para el movimiento del personaje son 4, uno por cada punto cardinal. No obstante en la mayoría de videojuegos el movimiento del personaje no está delimitado en estas cuatro direcciones, si no se que se puede mover de manera diagonal en el espacio, y este movimiento, en vez de ser representado por una animación en concreto, puede llevarse a cabo fundiendo dos o tres animaciones de movimiento en una sola.

5.8. NAVEGACIÓN Y PATHFINDING

Unity incorpora un sistema de navegación muy potente que permite a los objetos de la escena moverse de manera inteligente sobre el escenario. Esto se consigue creando una malla de navegación que cubre todos los objetos estáticos para navegación del juego de acuerdo a los parámetros designados en la ventana Navigation (Figura 34) a la hora de crear esta malla. Entre ellos se encuentran cuán cerca puede desplazarse el objeto de las paredes o los bordes de otros objetos, a qué altura se puede mover el objeto y cuán alto puede subir desde una superficie a otra. Con esta maya de navegación, cuando se ordena al objeto que se desplace de un punto a otro, este detecta el camino más corto entre ambos puntos y lo recorre.

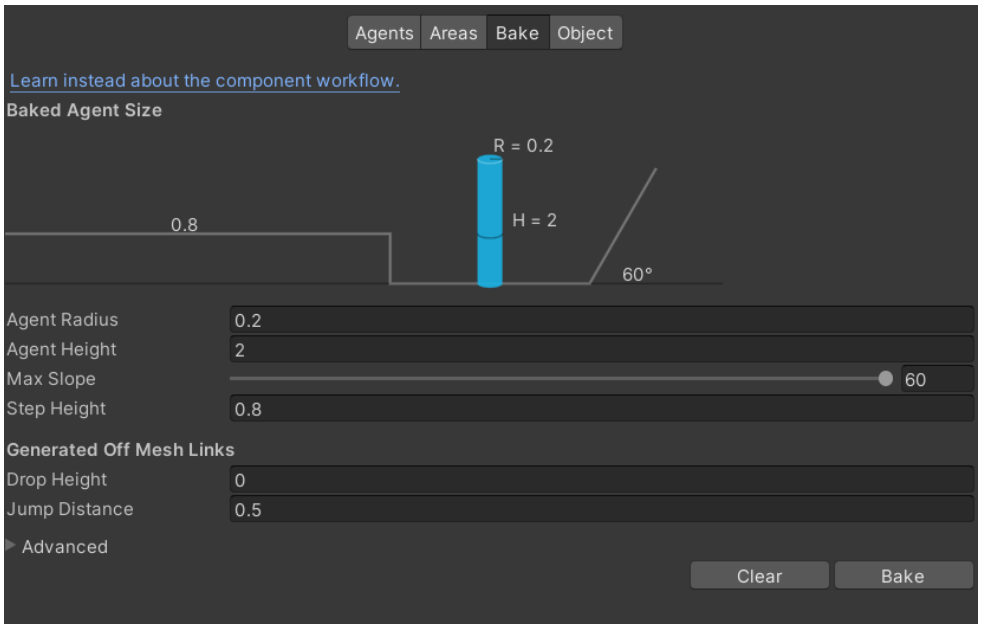


Figura 34: Ventana Navigation en Unity

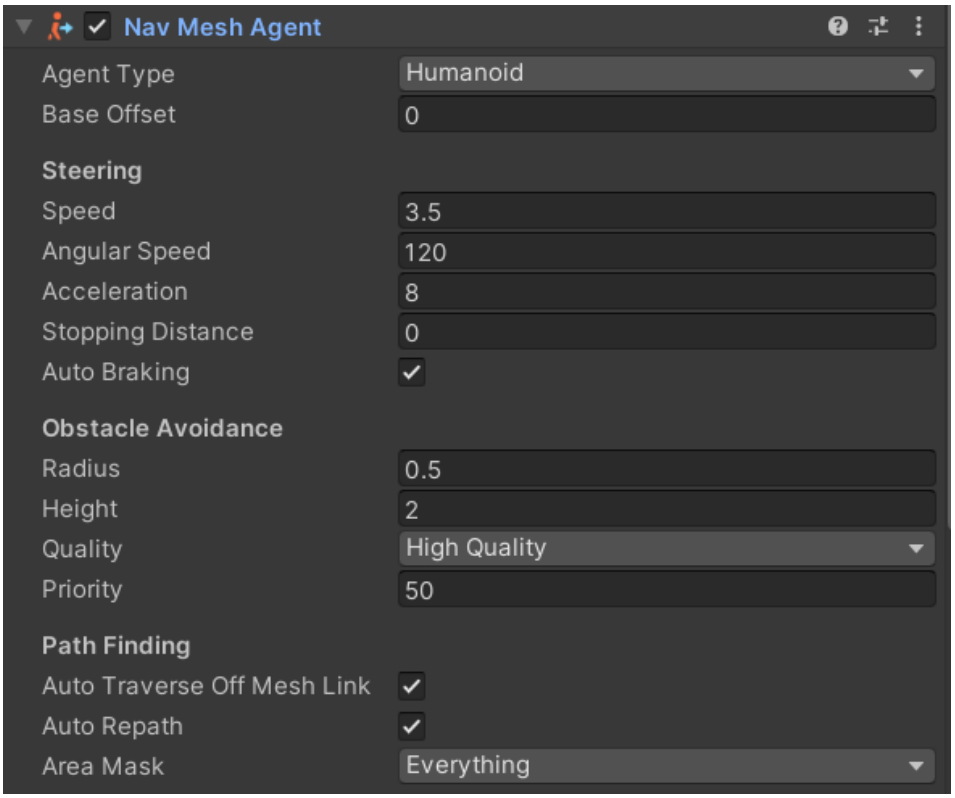


Figura 35: Componente Nav Mesh Agent

Dentro del objeto (o agente a partir de ahora) el cuál se quiere dotar de inteligencia para desplazarse se puede incluir una componente Nav Mesh Agent que permite definir magnitudes como velocidad de desplazamiento o aceleración, para modificar la manera en la que se mueve por la maya creada. Del mismo modo, se pueden crear obstáculos que deban ser evitados mientras se desplaza el agente de manera similar a si tuvieran un Collider y estos objetos no estuviesen marcados como navegables.

Por otro lado, se pueden crear pequeños atajos fuera de la maya de navegación entre dos puntos concretos utilizando Off-mesh Links. Estos componentes permiten por ejemplo un salto para pasar de un lado de un muro al otro o para saltar desde un terreno a una altura muy superior a otro.

5.8.1. Pathfinding Cost y Algoritmo de búsqueda A*

Ya que la maya de navegación creada a través del sistema de navegación de Unity es una maya compuesta de polígonos, el camino más corto que se calcula entre el origen y el destino es la conexión de estos puntos o nodos de cada polígono que convergen en la distancia más corta. El camino entre estos nodos tiene un coste asociado y dicho coste viene marcado por el área en el que se encuentran. Unity permite designar diferentes costes a cada área de navegación a razón de que un área con coste 3 es hasta tres veces más difícil de navegar que una de coste 1.

Estos costes asociados son utilizados junto al algoritmo de búsqueda A* para calcular el camino más corto. A* se utiliza para calcular el camino de menor coste entre un nodo origen y destino siempre y cuando se cumplan ciertas condiciones.

Este algoritmo surge a mediados del siglo 20 como respuesta a los algoritmos voraces que se utilizaban en la época y que solo tenían en cuenta la función heurística para calcular el camino más corto entre dos nodos. El problema de tomar exclusivamente la función heurística es que esta podía indicar el camino más corto pero no el de coste más bajo. También había un problema similar cuando se utilizaban algoritmos que solo tenían en cuenta el valor real de

desplazarse de un nodo a otro, haciendo que en ocasiones se utilizase un coste mayor debido a movimientos extra para alcanzar la solución. [38]

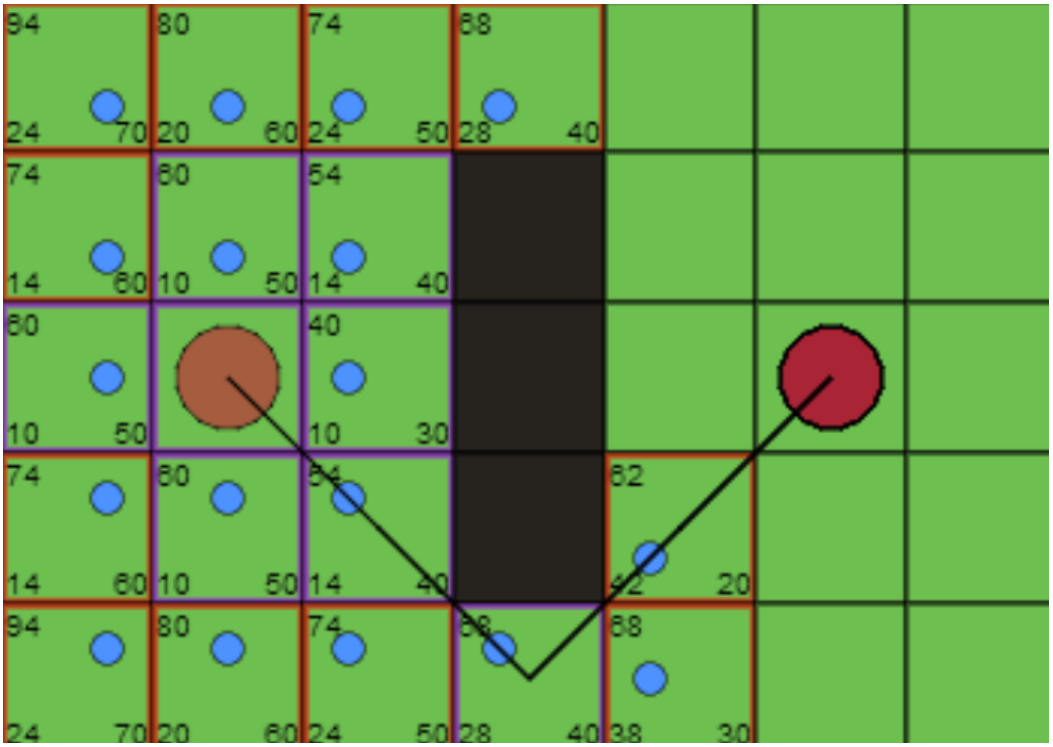


*Figura 36: Resultado de ejemplo interactivo del algoritmo A**

Con el algoritmo A* se fusionan ambas aproximaciones. La función de evaluación es f(n) = g(n) + h'(n), siendo h'(n) el valor heurístico del nodo que se está evaluando hasta el nodo final y g(n) siendo el coste real del camino recorrido hasta llegar al nodo n. El recorrido del algoritmo es simple. Se utilizan dos estructuras de datos auxiliares que se denominan open y closed. En open, que posee una estructura de cola de prioridad, se guardan los hijos nodos del nodo que se está evaluando pasándolos por la función f(n). En closed se guardan los nodos ya evaluados.

Siempre se evalúa el nodo de la lista open con el menor valor f(n), que en este caso al ser una cola de prioridad siempre es el elemento en primer lugar. Cuando se evalúa el nodo destino, este se añade a closed y termina el algoritmo. Al tener guardada también el nodo de padre de cada nodo, se puede realizar el camino inverso desde el nodo destino al nodo inicial y dicho camino es el de menor coste. *Aquí* se accede a una página que ofrece un ejemplo interactivo de la aplicación de este algoritmo. [40]

Este algoritmo siempre da con una solución en caso de existir, al igual que todos los algoritmos de búsqueda en amplitud. No obstante, para que se pueda garantizar que devuelve la solución óptima, es decir, el camino más corto real, la función heurística h(n) debe ser heurística admisible. Esto quiere decir que no debe sobrestimar el coste real de alcanzar el nodo n.

5.9. PLANES

En la actualidad Unity ofrece un gran catálogo de planes adaptados a las necesidades del usuario o entidad que quiere desarrollar con su tecnología [42]. Los más destacados son:

5.9.1. Unity Personal

La versión personal está destinada a usuarios individuales aficionados que quieran adentrarse en el mundo de desarrollo de videojuegos y, como tal, no estén dispuestos a pagar por un software que no saben si van a utilizar de manera prolongada o profesional.

También se destina a empresas pequeñas y el plan sigue siendo totalmente gratuito para estas si no superan los 100.000 dólares de ingresos en el último año. En ese caso se obliga a la empresa a adquirir alguno de los otros planes de pago disponibles de Unity.

Este plan es más que suficiente para desarrollar un videojuego con una calidad aceptable, ya que ofrece todas las herramientas básicas para la creación de un videojuego en 3D y 2D. Sin embargo y como es lógico, existen ciertos complementos que no recoge, como el acceso al código fuente de Unity, paquetes de assets de gama alta, análisis de operaciones elevado o el soporte técnico.

5.9.2. Unity Plus

Este plan está destinado a entidades o usuarios que obtengan ingresos superiores a 100.00 e inferiores a 200.000 dólares y ofrece a grandes rasgos mayor funcionalidad y recursos.

A diferencia del anterior este incluye todas las herramientas de colaboración que se quiera, operar con Cloud Diagnostics Advanced, qué básicamente recoge de manera automática errores y excepciones encontradas en los videojuegos y también ayuda a recopilar informes de bugs directamente de los usuarios

5.9.3. Unity Pro

Unity Pro sube un escalón el nivel y ofrece un software de desarrollo prácticamente completo destinado a profesionales del sector para que desplieguen y hagan uso de todas las herramientas de desarrollo que ofrece Unity. Es obligatorio para aquellas entidades que superen los 200.000 dólares de ingresos en los últimos 12 meses

Para destacar, Unity Pro implementa los assets de alta gama y tiene la posibilidad de utilizar plataformas cerradas además de otras funcionalidades. También es posible acceder al código fuente y a la atención del soporte técnico, aunque es necesario pagar unos costes adicionales.

Existe un modo de acceder a todas las funcionalidades que ofrece este plan de manera gratuita con el plan Unity Student. Con este plan y una vez se haya acreditado que la persona que quiere obtener los beneficios del mismo es alguien mayor de 16 años, que forma parte de una institución educativa acreditada (como por ejemplo una universidad) y que da su consentimiento para que se recopile y procese información personal, se puede acceder a todos estos recursos.

5.9.4. Unity Enterprise

Este es el plan más completo y por tanto, destinado a grandes compañías especializadas en el desarrollo software de videojuegos o de cualquier otro contenido digital que quieran hacer uso de

todas las herramientas posibles. Al igual que el plan Pro, es obligatorio para las empresas cuyos ingresos rebasen los 200.000 dólares anuales, aunque a mayores, se exige que el número de puestos sea superior a 20.

Ofrece absolutamente todas las funcionalidades que Unity tiene para todos los ámbitos del desarrollo, aunque, si bien es cierto, es necesario pagar algún plus extra para obtener acceso al código fuente, herramientas específicas para el desarrollo de soluciones para cada sector de la industria y guías estratégicas minuciosas para ayudar en la fluidez y la innovación del desarrollo del producto.

	PERSONAL	PLUS	PROFESIONAL	ENTERPRISE
Plataforma básica de desarrollo en tiempo real de Unity				
Herramienta Bolt para scripting visual				
Personalización de la pantalla de inicio				
Integraciones con las herramientas de colaboración	1, a elección			
Unity Teams Advanced (3 puestos)				
Crea e implementa en plataformas cerradas				
Paquete de assets de arte de alta gama				
Capacidad de licencias de Build Server				
Acceso al código fuente				
Conjuntos de herramientas de soluciones				

específicas para cada sector de la industria				
Cloud Diagnostics Advanced				
Análisis básico				
Analytics: Exporta 50 GB al mes de datos sin procesar				
Soporte técnico y asistencia				
Administrador del éxito del cliente				
Acceso prioritario a Unity Success Advisors				
Fila con prioridad para servicio al cliente				
On-Demand Training subscriptions (3 per 20 Enterprise seats)				
Servicios para el éxito integrados				

Tabla 1: Funcionalidad de cada plan de Unity

6. Entorno de desarrollo Rider

Unity utiliza la plataforma de código abierto .NET de Microsoft en su núcleo. Microsoft .NET es una plataforma de aplicaciones o un framework que permite la creación y ejecución de aplicaciones multiplataforma de una manera mucho más ligera y sencilla. Básicamente ofrece soluciones más robustas y sencillas, a la vez que económicas, a la hora de desarrollar aplicaciones software. [43]

Esta plataforma surge en el año 2000 (aunque no fue hasta el año 2003 cuando se centró su uso en el desarrollo software) como competencia a la plataforma Java y como respuesta al creciente mercado de desarrollo en entornos web.

Unity basa su tecnología de scripting en este framework .NET, lo que convierte al lenguaje C# en un perfecto candidato para ser usado en el desarrollo de aplicaciones en este motor, ya que es el estandarizado por Microsoft en la plataforma .NET y además es el utilizado como ejemplo en toda la documentación de Unity.

Rider es un entorno de desarrollo integrado (o IDE) multiplataforma destinato a C# y que utiliza el framework .NET de la compañía JetBrains. Este entorno surge posteriormente al editor de texto Visual Studio Code, que es la otra gran herramienta utilizada en el desarrollo de videojuegos en Unity.

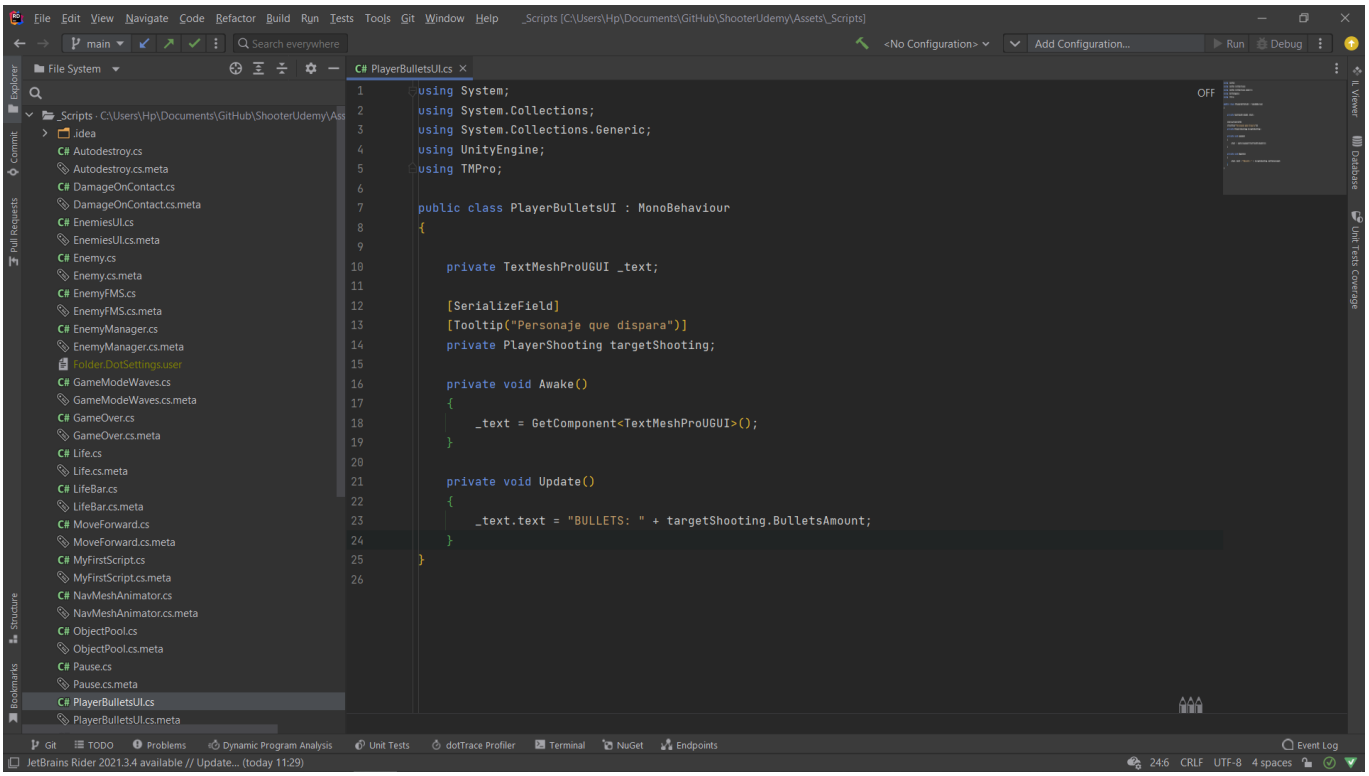


Figura 37: Ejemplo de interfaz de Jetbrains Rider

Aunque la popularidad de VSC es innegable y es el editor más utilizado y sobre el que más se comenta en internet, Rider presenta varias ventajas sobre este.

En cuanto a términos de productividad y optimización, Rider es claramente superior a Visual Studio o a Visual Studio Code. Es capaz de compilar código más rápidamente y de actualizar cambios realizados en el código en el editor de Unity, lo que lo hace ideal cuando los programas escalan y las soluciones a esos problemas son cada vez mayores. Además presenta herramientas para depurar errores mucho más precisas y la experiencia de usuario es más enriquecedora, ya que posee una interfaz clara y limpia

y permite una personalización de la misma muy profunda. Por otro lado también es claro ganador en términos de análisis de código y refactorización, siendo más sencillo eliminar código innecesario o “code smells” mientras se está desarrollando.

No obstante, Visual Studio Code ofrece un mayor abanico de programas adicionales y características para añadir que Rider, además de más tipos de plantillas para diferentes frameworks y lenguajes, ya que Rider solo puede trabajar con lenguajes que utilicen .NET como plataforma para compilar. Además es importante señalar que Rider es de pago, a diferencia de VSC. Sin embargo, existen periodos de prueba en los que se puede disfrutar de este IDE de manera gratuita, además de licencias para alumnos y profesores de estudios informáticos de un periodo de tiempo relativamente largo. [44]

Haciendo un balance general de las características específicas de cada programa y de las recomendaciones recibidas por parte del curso de desarrollo de videojuegos con Unity de la plataforma de aprendizaje Udemy, se decidió realizar el desarrollo con Rider.

En *esta* página se pueden realizar comparaciones entre distintos IDEs.

7. Planificación del proyecto

7.1. Características del proyecto

El proyecto consiste en un videojuego de Realidad Virtual para las gafas Oculus Quest 2. Los participantes de dicho proyecto son dos estudiantes de último año del Grado en Ingeniería en Diseño Industrial y Desarrollo del Producto, dos estudiantes de un Ciclo Formativo de Grado Superior de Animación 3D y un estudiante de último año del Grado en Ingeniería Informática (el autor de este trabajo).

También formará parte del proyecto el Ingeniero en Diseño Industrial y Desarrollo del Producto Pablo Gatón, el cuál es también CEO de Irzón, la empresa que lleva a cabo el proyecto y la cuál nos ha proporcionado el terreno donde se desarrollará el videojuego.

7.2. Metodología de desarrollo

Para el desarrollo del proyecto se ha elegido adoptar la metodología de desarrollo ágil SCRUM.

La metodología ágil surge en el año 2001 en contraposición a la metodología de desarrollo de proyectos software en cascada, que era la que predominaba en ese momento. Surge un manifiesto [45] con una serie de valores y principios que proponían otro enfoque de desarrollo y que se basaba en una serie de valores fundamentales que se resumen en poner los individuos y las interacciones por delante de los procesos y herramientas, que un software funcional es más importante que una documentación completa, que la colaboración contínua con el cliente es más relevante que la negociación contractual y que la adaptabilidad a los cambios debe estar por encima del seguimiento de un plan inicial.

Este manifiesto recoge con más detalle 12 principios que deben cumplirse para tener un desarrollo ágil en un proyecto software. Algunos de los más destacables son la entrega temprana y continua de software funcional al cliente, el buen recibimiento de requisitos cambiantes, la relación cara a cara entre los miembros del equipo, la simplicidad debe estar siempre presente y que la autoorganización del equipo y la motivación de cada individuo deben ser los pilares para un trabajo contínuo.

En el desarrollo en cascada, el ciclo de vida es prácticamente lineal, con unas fases de desarrollo muy concretas y separables unas de otras y con dificultades para volver a fases anteriores en el desarrollo. El ciclo de vida de la metodología ágil es iterativo e incremental. Esto quiere decir que es fácil adaptar el desarrollo del proyecto a cambios en los requisitos por parte del cliente y que también el tiempo de desarrollo de un software lo suficientemente funcional como para servir al cliente es relativamente corto y los incrementos realizados en ese software se producen de manera continuada, ofreciendo valor al cliente.

Dentro de la metodología ágil existen diferentes enfoques, pero el más conocido y el que se usará en este proyecto es la metodología ágil SCRUM. En la metodología SCRUM se tienen principalmente 3 roles

- **Product Owner**

  Es el encargado de hablar con los clientes (Stakeholders, ejecutivos, usuarios finales…) y recoger los requisitos del producto que hay que desarrollar en el Product Backlog. En este proyecto el Product Owner es Pablo Gatón, CEO de Irzón.

- **SCRUM Master**
  En segundo lugar, un SCRUM Master, que es el responsable de que los principios de la metodología SCRUM sean aplicados correctamente. Es una persona con gran experiencia en el desarrollo de proyectos que utilicen esta metodología y, a pesar de ser el jefe del proyecto, tiene el rol de facilitador del equipo. Debe además servir de moderador en el equipo y de solucionador de problemas. En este proyecto yo tomaré el rol de SCRUM Master.

- **Equipo de desarrolladores**
  Por último están los miembros del equipo de desarrollo. Un equipo de desarrollo en esta metodología está formado por entre 3 y 9 personas, con características y especialidades diferenciadas. Su trabajo principal es convertir lo anotado en el Product Backlog en incrementos de funcionalidad. En este trabajo también participa el SCRUM Máster.

  Como ya se ha dicho antes, el equipo debe ser autoorganizado y debe estar compuesto por personas que conozcan la metodología SCRUM y que sean comunicativos los unos con los otros. En este proyecto los desarrolladores son los estudiantes del Grado en Ingeniería en Diseño Industrial y Desarrollo del Producto, los otros dos miembros del equipo. Aunque yo sea el SCRUM Master, también formaré parte del equipo de desarrolladores.

El desarrollo comienza con la anotación de los requisitos en el Product Backlog por parte del Product Owner. Estos requisitos que aparecen recogidos se llaman Historias épicas y se encuentran subdivididos en Historias de usuario.

Las Historias épicas son requisitos generales que debe cumplir el software desarrollado. Las Historia de usuario son acciones concretas que el usuario debe ser capaz de realizar y que junto a otras Historias de usuario generan una funcionalidad recogida en una Historia épica.

En este proyecto se ha seguido una adaptación de la metodología SCRUM, acondicionando sus reuniones al ritmo de trabajo que teníamos. Se realizó una reunión inicial en la que el Product Owner mostró los requisitos deseados y a partir de ese momento, se tuvieron reuniones periódicas cada semana aproximadamente para repasar de nuevos los requisitos, redefinir aquellos que lo necesitasen y añadir o eliminar otros. Además prácticamente todos los días se tenían reuniones de entre 15 y 30 minutos para comprobar cómo estaba yendo el desarrollo de la Historia en la que se estuviera trabajando en ese momento.

7.3. Historias de usuario

A continuación se detallan las Historias épicas incluidas en el Product Backlog y subdivididas en Historias de usuario. También se detalla el tiempo estimado para su realización, la persona encargada de su realización y una prioridad de realización (número entre 1 y 10). El SCRUM Máster es Pablo Gatón y los desarrolladores 1, 2 son los estudiantes del Grado en Ingeniería en Diseño Industrial y Desarrollo del Producto y el desarrollador 3 yo. El esquema viene reflejado en la Tabla 11.

Número de historia		
Descripción		
Persona encargada	**Prioridad**	**Estimación**

Tabla 2: Tabla ejemplo de Historia dentro del Product Backlog

01		
Descripción: El escenario del videojuego será una maqueta realista a escala 200:1 del Castillo de la Mota de Medina del Campo.		
Persona encargada: SCRUM Máster	**Prioridad :** 5	**Estimación :** ya hecha

02		
Descripción: El castillo estará dividido en diferentes puntos por los que se pueden mover las unidades atacantes y defensivas. Las unidades se moverán de punto a punto. Las unidades defensivas atacarán desde determinados puntos del castillo		
Persona encargada: Desarrollador 2	**Prioridad:** 10	**Estimación:** 3 días

03		
Descripción: La interfaz dentro del juego estará compuesta por un menú principal que derive en otras opciones. Existirá un menú para cambiar los ajustes del juego. Existirá también un menú en el que el usuario podrá elegir el tipo de soldado que quiere colocar en el castillo. También habrá un menú para seleccionar la mejora que quiera comprar.		
Persona encargada: Desarrollador 1	**Prioridad:** 7	**Estimación:** 7 días

04
Descripción: El movimiento de las unidades atacantes será aleatorio y de avance hacia la torre del homenaje del castillo. Fuera de la primera muralla el movimiento será siempre hacia delante, a menos que no pueda avanzar a ningún punto de delante, en cuyo caso será horizontal. Las unidades atacantes intentarán penetrar el castillo escalando la muralla exterior cuando ésta esté parcialmente destruida o entrando por la puerta exterior cuando haya sido destruida. El movimiento de las unidades atacantes dentro de la primera muralla será siempre en horizontal. La unidad buscará escalar la segunda muralla cuando alguna parte haya sido destruida o entrar por la puerta del patio de armas cuando haya sido destruida. Si esa unidad ha escalado la muralla, deberá priorizar bajar de las almenas primero.

Las unidades atacantes tendrán como prioridad atacar a los defensores que se encuentren dentro del rango de ataque.

El ataque de una unidad estará condicionado por su tipo y por el tipo de las unidades enemigas que tenga a rango.

División en US:

☐

Persona encargada: Desarrollador 3	**Prioridad:** 9	**Estimación:** 7 días

05		
Descripción: Toda la información necesaria para que el juego fluya que pueda estar precalculada antes de que el jugador comience a jugar deberá cargarse antes.		
Persona encargada: Desarrollador 3	**Prioridad:** 7	**Estimación:** 3 días

06		
Descripción: El usuario podrá seleccionar las unidades defensoras del castillo y ordenar que se muevan a otro punto de ataque siempre y cuando ese punto de ataque esté disponible para esa unidad. Las unidades seguirán el camino más corto posible entre los puntos de ataque.		
Persona encargada: Desarrollador 3	**Prioridad:** 7	**Estimación:** 5 días.

07		
Descripción: El usuario podrá seleccionar qué unidad desea comprar y posicionar en el castillo y podrá posicionarla de manera sencilla en él, si tiene los recursos necesarios para hacerlo.		
Persona encargada: Desarrollador 3	**Prioridad:** 6	**Estimación:** 5 días

08		
Descripción: El usuario podrá acceder a descripciones detalladas con imágenes de las armas que portan las unidades defensoras y de las mejoras que puede adquirir.		

Persona encargada: Desarrollador 2	**Prioridad:** 6	**Estimación:** 4 días

09		
Descripción: El juego tendrá diferentes bandas sonoras de acuerdo al momento de la partida en el que se encuentre y a la situación de la misma.		
Persona encargada: Desarrollador 1	**Prioridad:** 7	**Estimación:** 3 días

10		
Descripción: El videojuego tendrá un sistema de oleadas. Cada oleada será más difícil que la anterior, bien porque contiene más enemigos o bien porque contiene enemigos más fuertes.		
Persona encargada: SCRUM Master y Desarrolladores 1, 2 y 3	**Prioridad:** 9	**Estimación:** 6 días

11		
Descripción: Cada arma tendrá unas características determinadas que influyen en el daño que puedan llegar a realizar, desde dónde puede disparar el personaje que la porta y con qué frecuencia pueden hacerlo y a qué velocidad puede moverse por el mapa.		
Persona encargada: SCRUM Master y Desarrolladores 1, 2 y 3	**Prioridad:** 5	**Estimación:** 2 días

12		
Descripción: El juego terminará cuando el jugador supere todas las oleadas. Los atacantes y los defensores actúan únicamente si el juego se encuentra en fase de batalla. Entre oleadas los enemigos no aparecen y el jugador no puede reposicionar las tropas con las que ya cuenta.		
Persona encargada: Desarrollador 3	**Prioridad:** 5	**Estimación:** 2 días

7.4. Planificación inicial

La reunión de arranque del proyecto y del primer sprint tuvo lugar el día 28 de marzo de 2022.
En esta reunión se presentó el Product Backlog con las historias de usuario que se deben desarrollar. Es importante destacar que el Product Backlog es modificado a lo largo del periodo de desarrollo para ajustarse a nuevas especificaciones de la aplicación.

En esta reunión se determinó una división aproximada de los Sprints que tendría el desarrollo del proyecto.

- **Sprint 1:** 28/03 – 04/04. 6 días (42 horas)

  Durante el Sprint 1 se realizará la historia 05. Se divide en las siguientes tareas:

  - Crear una base de datos en la nube que sirva como punto de guardado de los datos del juego y del escenario
  - Crear scripts fuera del proyecto que recojan los datos necesarios del juego y los guarden en la base de datos.
  - Crear scripts dentro del juego que al comienzo de la partida carguen esos datos.

- **Sprint 2:** 05/04 – 19/04. 10 días (70 horas)

  Durante el Sprint 2 se realizará la historia 04. Se divide en las siguientes tareas:

  - Realizar el diseño del cambio de estado de atacar a moverse y viceversa de la unidad atacante.
  - Diseñar el movimiento hacia delante y horizontal de fuera de la primera muralla.
  - Diseñar el comportamiento de la unidad para subir la muralla
  - Diseñar el camino que debe seguir la unidad atacante para bajar de las almenas a dentro de la primera muralla.
  - Diseñar el movimiento horizontal de dentro de la primera muralla.

- **Sprint 3:** 19/05 – 29/04. 12 días (84 horas)

  Durante el Sprint 3 se realizará la historia 06. Se divide en las siguientes tareas:

  - Establecer un sistema de guardado de distancias entre cada punto de ataque a puntos clave del mapa, para establecer un mapa de distancias.
  - Programar el guardado de estas distancias clave en el script de guardado de datos.
  - Dividir los puntos de ataque del castillo de acuerdo al tipo de unidad que puede haber en ellos
  - Programar el camino de puntos más pequeño entre dos puntos de ataque teniendo en cuenta las almenas y el cambio entre pisos
  - Mostrar con una línea el camino de puntos más corto que se debe recorrer para llegar de un punto a otro en tiempo real.

- **Sprint 4:** 29/04 – 11/05. 10 días (70 horas)

  Durante el Sprint 4 se realizará la historia 11, 12 y . Se divide en las siguientes tareas:

  - Crear un gestor de oleadas que administre lo relativo a las mismas
  - Programar la creación de una oleada, con todas sus características.
  - Programar la aparición de los enemigos

  - Crear un gestor de partida que controle el transcurso del juego

- **Sprint 5:** 11/05 – 17/05. 5 días (35 horas)

  Durante el Sprint 5 se realizará la historia 07. Se divide en las siguientes tareas:

  - Asignar botones que funcionen como listeners para detectar qué opción exactamente selecciona el usuario del menú.
  - Programar la acción o acciones que se llevan a cabo cuando el usuario selecciona una opción.
  - Añadir restricciones de número de oleada y dinero disponible a las opciones del menú que puede seleccionar el usuario

7.5. Identificación de riesgos

La identificación de los riesgos que pueden suceder en un proyecto es uno de los pasos fundamentales en el desarrollo de software.

Es necesario identificar y calificar de manera cuantitativa el probabilidad de ocurrencia y el impacto sobre el desarrollo en caso de que estos riesgos se materialicen, además de establecer planes para evitar que aparezcan y para enfrentarse a ellos en caso de tenerlos presentes.

La probabilidad (Tabla 2) e impacto (Tabla 3) de cada riesgo se medirá de manera cuantitativa con un número entre el 1 y el 10, 1 siendo lo más leve y 10 lo más peligroso. Con estos valores se calculará la exposición de riesgo de cada riesgo detectado y se rellenará una matriz de riesgo representativa de la importancia de actuación sobre todos los riesgos detectados. En el caso de estar entre dos números, el inferior no se incluye.

Nivel de Probabilidad	**Rango**
Alta	Más de 8 puntos
Media-Alta	Entre 6 y 8 puntos
Media	Entre 4 y 6 puntos
Medio-Baja	Entre 2 y 4 puntos
Baja	Menos de 2 puntos

Tabla 3: Descriptores cualitativos de riesgo asociado

Nivel de Impacto	Rango
Alto	Más de 5 puntos
Significativo	4 puntos
Moderado	Entre 2 y 4 puntos
Despreciable	1 punto

Tabla 4: Descriptores cualitativos de impacto asociado

A continuación se detalla una tabla con los riesgos identificados que pueden suceder en el desarrollo del proyecto.

Cada riesgo tiene un número que sirve como identificador, una breve descripción del problema, una probabilidad de que suceda y el impacto que supondría y por último, una o varias medidas para evitar la aparición del riesgo (si es posible evitar su aparición).

Riesgo	Descripción	Probabilidad	Impacto	Planes de prevención
1	Se han definido requisitos de manera errónea y esto desemboca en una nueva definición de dichos requisitos y la modificación del trabajo ya realizado	8	8	1. Realizar un estudio exhaustivo de los requisitos funcionales y no funcionales de la aplicación 2. Revisar los requisitos sobre los que se va a desarrollar antes de comenzar el desarrollo de esa parte
2	Existen requisitos del proyecto que no han sido descritos durante las reuniones y esto desemboca en un desarrollo de proyecto inadecuado	8	9	1. Realizar un estudio exhaustivo de los requisitos funcionales y no funcionales de la aplicación 2. Organizar los requisitos en función del área de desarrollo que afectan.
3	La herramienta usada para trabajar de manera colaborativa no se encuentra operativa	4	2	1. Tener al menos una herramienta colaborativa de repuesto. 2. Establecer copias físicas de seguridad periódicas
4	La falta de conocimiento sobre el motor Unity ralentiza el proceso de	7	9	1. Tener acceso a tutorial de Unity de manera intermitente y también a libros de formación

	desarrollo			
5	La falta de conocimiento en el área de programación y en la programación en C# ralentiza el proceso de desarrollo	8	7	1. Tener acceso a tutoriales de programación en C# (orientada o no al desarrollo de Unity) de manera intermitente y a libros de formación.
6	El uso del motor Unity consume muchos recursos en un ordenador que no puede prescindir de ellos y esto genera congelamientos periódicos.	4	5	1. Uso de ordenadores con suficiente capacidad como para correr Unity y más programas a la vez.
7	Alguno de los miembros cae enfermo y no puede trabajar durante un periodo de tiempo	3	2	
8	Los miembros del equipo no trabajan bien juntos. No existe un ambiente de equipo	2	7	1. Realizar actividades o pequeños momentos de reunión fuera del trabajo para aumentar la cohesión del equipo
9	La herramienta de comunicación on-line no funciona.	3	3	1. Establecer una o varias alternativas de comunicación on-line.
10	Fallo en el suministro de energía en los ordenadores en los que se trabaja, perdiendo el proceso no guardado.	2	5	1. Realizar copias de seguridad físicas periódicas. 2. Establecer opciones de auto-guardado en los programas usados en el desarrollo.
11	El tiempo de desarrollo de alguna tarea es mayor que el estimado.	7	6	1. Destinar más recursos a dicha actividad (personas) 2. Posponer o desechar tareas futuras que no sean críticas.
12	Se mezclan archivos propios del ordenador de cada miembro del grupo en el repositorio remoto, ocasionando	6	5	1. Establecer las carpetas y archivos que no deben subirse al repositorio común y anotarlas en un archivo automatizado que

	conflictos.			impida la subida de dichas carpetas y archivos cuando se actualiza el repositorio on-line
13	El desarrollo de la interfaz de usuario es errónea y no intuitiva	3	8	1. Establecer en reuniones previas las interfaces necesarias, sus características y sus componentes necesarias.

Tabla 5: Riesgos asociados al proyecto

Para continuar se calculará la exposición al riesgo (Risk Exposure = potential damage x probability of ocurrence) en la tabla 6.

Riesgo	Probabilidad	Impacto	Exposición al Riesgo
R1	8	8	64
R2	8	9	72
R3	4	2	8
R4	7	9	63
R5	8	7	56
R6	4	5	20
R7	3	2	6
R8	2	7	14
R9	3	3	9
R10	2	5	10
R11	7	6	42
R12	6	5	30
R13	4	8	32

Tabla 6: Cálculo de la Exposición al Riesgo.

En siguiente lugar, se calcula la matriz de riesgo de acuerdo a los valores recogidos en la tabla 6.

Alto			**R5**	**R1, R2, R4**	
Significativo		**R12**	**R11**		
Moderado	**R8, R10**	**R6, R13**			
Despreciable	**R3, R7, R9**				
Impacto / Probabilidad	**Baja**	**Media-Baja**	**Media**	**Media-Alta**	**Alta**

Tabla 7: Matriz de probabilidad de impacto

En la Tabla 7, las casillas marcadas en verde recogen los riesgos que se pueden ignorar debido a su mínima importancia. En la amarilla se definen los riesgos para los cuáles es necesario tener presentes las medidas cautelares necesarias para abordarlos en caso de que pasen a alguna celda de rojo. En las celdas rojas aparecen los riesgos sobre los cuáles hay que tomar medidas inmediatamente. En negro están los riesgos que hacen inviable el proyecto.

Los riesgos que se encuentran por encima de la línea de tolerancia son aquellos que no pueden permitirse en el proyecto y debe llevarse a cabo su solución de manera inmediata. La línea de tolerancia no corresponde a una fórmula exacta. En este caso se dibuja debajo de las casillas de la matriz marcadas de color rojo.

De acuerdo a esta matriz, el R1, el R2, el R4 y el R5 son riesgos críticos que hay que abordar de forma urgente.

Por último, en la tabla 8 se detalla las medidas de contingencias que se llevarían a cabo para cada requisito en caso de su aparición, que se derivan en parte de las medidas de prevención anteriormente descritas.

Riesgo	**Descripción Breve**	**Medidas de Contingencia**
R1	Mala definición de requisitos	Se realizará de nuevo un estudio desde el inicio de los requisitos funcionales y no funcionales que debe cumplir el proyecto para evitar nuevas apariciones del riesgo. Se reformulará el requisito con el aprobado general de los miembros del proyecto y en última instancia, si es muy complejo, se eliminará o se modificará completamente.
R2	Falta de requisitos	Se repasará de manera exhaustiva los requisitos que afectan al mismo área que el requisito que ha desembocado el riesgo y se comprobará que dicho área queda plenamente definida
R4	Inexperiencia con Unity	Se invertirá más dinero y tiempo en cursos de formación en Unity. Se considerará contratar desarrolladores cualificadas para ofrecer formación presencial
R5	Inexperiencia con C#	Se considerará realizar una formación intensiva o bootcamp para C# dentro de la empresa. Se ofrecerán

		cuentas premium en foros de Microsoft para resolver dudas en el menor tiempo posible.

Tabla 8: Medidas de contingencia

7.6. Desarrollo del proyecto

Sprint 1

El sprint 1 comenzó el día 28 de Marzo y terminó el día 8 de Abril. Ha habido una desviación de 4 días con respecto a la planificación inicial del sprint 1.

Los principales motivos de dicha desviación han sido que el tiempo de desarrollo para realizar la Historia 5 ha sido mayor que el tiempo planeado (Riesgo 11) y que se ha hecho patente la falta de conocimientos de programación en C# (Riesgo 5).

Sprint 2

El sprint 2 comenzó el día 8 de Abril y terminó el día 29 de Abril. Se ha comenzado 4 días más tarde y se ha finalizado 10 días más tarde de lo planeado.

El principal motivo de dicha desviación ha sido que el tiempo de desarrollo para la Historia 4 ha sido mayor que el esperado (Riesgo 11). Además algunos requisitos han tenido que modificarse durante el sprint al haberse definido de manera errónea (Riesgo 1).

Sprint 3

El sprint 3 comenzó el día 29 de Abril y terminó el día 20 de Mayo. Se ha comenzado 10 días más tarde y se ha finalizado 22 días más tarde de lo planeado.

Los principales motivos de la desviación han sido que había que corregir los fallos de la Historia 4 que se desarrolló en el anterior Sprint, que se ha tardado más de lo previsto en desarrollar la Historia 6 y que algunos requisitos no estaban previamente definidos, por lo que se ha dedicado parte del tiempo a ello (Riesgo 2).

Sprint 4

El sprint 4 comenzó el día 20 de Mayo y terminó el día 15 de Julio. Se ha comenzado 22 días más tarde y se ha terminado más de 1 mes más tarde de lo planeado.

El principal motivo del retraso ha sido que había muchos errores que corregir de la historia 6 desarrollada en el Sprint 3. Además se han escogido muchas otras dos historias a mayores que desarrollar en el Sprint y eso ha alargado el desarrollo.

Sprint 5

El sprint 5 comenzó el día 15 de Julio y terminó el 5 de Junio. Se ha comenzado un més más tarde y se ha terminado 18 días más tarde.

Los principales motivos de dicha desviación han sido las desviaciones que se arrastraban de sprints anteriores y que ha habido que dedicar tiempo a redactar la documentación del proyecto.

7.7. Recursos hardware del proyecto

Ordenadores de sobremesa

Los ordenadores necesarios para realizar un proyecto de desarrollo software han de tener un procesador con unas prestaciones mínimas como para permitir ejecutar varios programas relativamente pesados de manera fluida . Es especialmente importante que incluyan una tarjeta gráfica dedicada, ya que se utilizarán programas donde el tratamiento de la imagen es importante. En la Tabla 9 se muestra la configuración de la máquina usada.

Procesador	Intel Core i5-3740
Memoria RAM	8 GB
Disco duro	HDD de 500 GB
SO	Windows 10 PRO
Tarjeta gráfica	GeForce 1050ti

Tabla 9: Principales componentes de los ordenadores de sobremesa

Pantallas

Uno de los factores que puede hacer más fluido el desarrollo es el de una doble pantalla por cada torre de ordenador. Con dos monitores por ordenador, se puede trabajar de una manera mucho más rápida, ya que se tiene más información presente a la vista y se puede pasar de trabajar con un programa a trabajar con otro sin perder de vista los datos del primero. En este caso, el ejemplo más claro sería tener una pantalla abierta el motor de Unity y en otra el entorno de desarrollo utilizado o simplemente Internet.

Las medidas de las pantallas deberían ser lo suficientemente grandes como para poder ver sin dificultad todos los detalles que ofrece la interfaz de Unity. Monitores estándar Full HD 1920 x 1080 de cualquier marca son suficientes.

En este caso las pantallas utilizadas son unas LG 25UM58-P de 25 pulgadas (62 centímetros) con Panel IPS, 75Hz, 250nit, proporciones 21:9 y con HDMI incorporado.

Ratones

Los ratones necesarios para trabajar con el ordenador son el elemento menos importante sobre el que prestar atención, pero es ideal que cumplan un mínimo de características para que su uso no perjudique el desarrollo.

En este caso se va a utilizar ratones VicTsing inalámbricos 2.4G con 2400 DPI y 6 botones con Nano receptor.

Gafas de realidad virtual

Las gafas de realidad virtual son necesarias prácticamente desde el inicio del proyecto, ya que se deben realizar pruebas de manera iterativa hasta que se obtenga el producto final. Las gafas que se usarán son las Oculus Quest 2 y el kit de compra ofrecen unas gafas, un cargador y un adaptador de corriente, dos controladores Touch con pilas triple A incluidas y un separador para las gafas.

7.8. Recursos Software

- **Unity:** el software utilizado de Unity consiste en Unity Hub, la aplicación de escritorio para administrar los proyectos Unity y la versión del Editor de Unity 2020.3.31f1
- **Jetbrains Rider 2021.3.1:** utilizado por mi para la implementación de la lógica del videojuego
- **Visual Studio Code:** para resolver determinados problemas en el desarrollo del videojuego
- **Google Drive:** para la redacción de la memoria del proyecto y la puesta en común del trabajo de los diferentes miembros del equipo
- **dokuwiki:** herramienta online para el desarrollo de proyectos para empresas para guardar la información relativa al desarrollo del videojuego
- **Github:** herramienta de desarrollo colaborativo de código libre en la nube basada en Git
- **GitHub Desktop:** aplicación de escritorio que permite al usuario manejar GitHub usando una interfaz de usuario en vez de una terminal.
- **Catia:** programa informático para el diseño y fabricación industrial

7.9. Identificación de costes

Cuando se habla de desarrollar un proyecto, ya sea software o no, es de extrema importancia realizar un buen análisis de su coste, teniendo en cuenta todos los factores que intervienen en el proyecto, como proveedores, distribuidores, diseñadores, artistas, desarrolladores, espacio físico, material de trabajo y un largo etcétera.

Al ser un proyecto de desarrollo de un videojuego, existen ciertas peculiaridades que se deben tener en mente cuando se realiza esta identificación de los costes.

En este proyecto, el espacio físico requerido era mínimo ya que eramos pocos miembros en el equipo de desarrollo, por lo que las oficinas donde se ha realizado parte del trabajo eran pequeñas y poco costosas. Además parte del trabajo ha sido realizado de manera remota. Por otro lado, los recursos materiales utilizados son principalmente ordenadores, pantallas, consolas para realizar pruebas y tablets, además de los recursos que el personal del departamento artístico necesite. Por tanto, el balance del coste de los recursos físicos puede variar bastante dependiendo de la calidad que se necesite para desarrollar el producto. En muchas ocasiones no será necesario contar con equipos altamente tecnológicos, y por ende, costosos,x si el desarrollo puede realizarse con equipos más modestos.

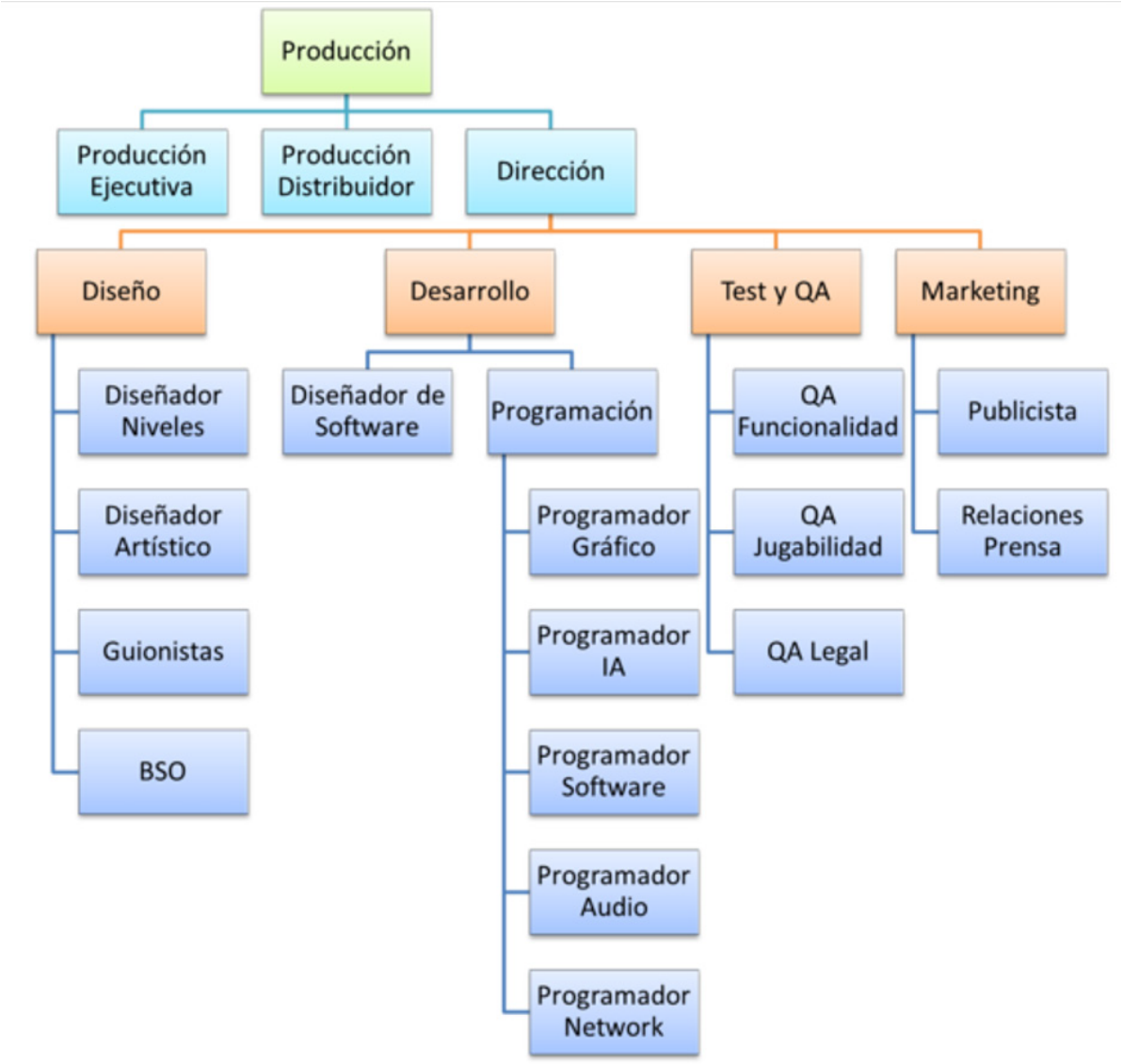


Figura 38: Ejemplo de diagrama de organización de departamentos en el desarrollo de un videojuego

Aparte de todo esto, es muy importante tener presente el apartado de marketing y distribución del producto a la hora de dividir costes. A pesar de que no forman parte del desarrollo del producto, su trabajo puede ser decisivo a la hora de atraer a clientes y posicionar el producto, con lo que un departamento de marketing con unos empleados cualificados y con unos buenos recursos es también necesario.

Por último, es importante considerar los costes asociados a licencias de uso de software de terceros. En muchas ocasiones, los programas que se usan para desarrollar un proyecto software ofrecen versiones gratuitas bastante completas, aunque si se quiere acceder a más prestaciones o si el volumen de ingresos es superior a un umbral, es necesario pagar una cuota a la empresa que provee ese software.

En este proyecto en concreto los gastos se dividirán de la siguiente forma. La dedicación del proyecto ha sido de 3 meses y una semana y media. La jornada de trabajo ha sido de 8 horas a la semana 5 días a la semana, de Lunes a Viernes. Se toma en cuenta que los meses tienen 4 semanas. En la tabla 10 se detalla el sueldo medio mensual que recibe cada miembro del equipo y el sueldo total correspondiente al trabajo realizado en el proyecto [46].

Empleado	**Coste mensual**	**Total**
Ingeniero en Diseño Industrial y Desarrollo del Producto (+5 años de experiencia)	2.575,5 €	8.692,31 €
Ingeniero en Diseño Industrial y Desarrollo del Producto (sin experiencia)	1.666 €	5.633,75 €
Ingeniero en Ingeniería Informática (sin experiencia)	1.704 €	5.751 €
Animador 3D	1.625 €	5.484,37 €

Tabla 10: Coste de cada empleado

Por otro lado también es importante estimar el gasto en equipo informático necesario en el proceso de desarrollo. En este caso el equipo necesario ya se encontraba en la empresa, por lo que el coste de adquirirlo ha sido nulo.

A mayores hay que sumar los gastos básicos de espacio de trabajo como luz, conexión a internet, ventilación y el coste asociado al uso de un espacio físico como puede ser una oficina.

Los demás recursos que se usarán para el desarrollo del proyecto son gratuitos. Esto incluye material gratuito de la tienda de Unity, el entorno de desarrollo Rider que ofrece hasta 1 año de uso gratis si se proporciona una licencia académica.

De acuerdo a lo expuesto anteriormente el coste total del proyecto asciende a **25.561, 44 €**

8. Desarrollo detallado del proyecto

En este apartado se describe cómo ha sido el desarrollo del proyecto. Principalmente se muestra el orden en el que se han abordado las historias del Product Backlog y se detalla cómo han sido resueltas. Los diagramas correspondientes se recogen en los apéndices 2 al 6.

Sprint 1

- **Requisitos**

  Historia 05

- **Tareas**

  - Crear una base de datos en la nube que sirva como punto de guardado de los datos del juego y del escenario
  - Crear scripts fuera del proyecto que recojan los datos necesarios del juego y los guarden en la base de datos.
  - Crear scripts dentro del juego que al comienzo de la partida carguen esos datos.

- **Implementación**

  Tras examinar diferentes opciones, se escoge utilizar Firebase, una plataforma destinada al desarrollo de aplicaciones web y móviles que posee una base de datos no relacional muy potente, además de brindar herramientas a los desarrolladores para la administración de sus proyectos.

  Dentro de Firebase se ha decidido utilizar el servicio Firebase Cloud Firestore. Firebase Cloud Firestore es una base de datos en tiempo real organizada en una estructura JSON compuesta por colecciones y documentos. Al brindar una sincronización en tiempo real, cualquier usuario conectado puede acceder a la información deseada de manera inmediata y modificar y crear documentos y diccionarios.
  Para la puesta en marcha de este servicio en la aplicación, es necesario instalar un SDK en Unity.

  A mayores, también se implementa el servicio Firebase Auth, que incorpora servicios de autentificación del usuario a través de proveedores de inicio de sesión como Facebook, Twitter o Google, así como incluir un sistema de administración de usuario y contraseña que se puede almacenar en la propia base de datos de Firebase.

  Una vez se logra introducir de manera correcta los datos en la base de datos, se programan los scripts que tienen que conectarse para recoger los datos e introducirlos en el juego. Para separar por completo la lógica del negocio de la lógica de acceso a datos se usa el patrón DAO (Data Access Object) en combinación con el patrón DTO (Data Transfer Object).

  Para hacer una breve explicación, el patrón DAO se implementa creando una clase que contiene las operaciones de acceso a la base de datos. Cuando el cliente quiere acceder a los datos, crea un objeto DAO e invoca una llamada a este objeto de acuerdo a qué operación

CRUD quiere realizar (Create, Read, Update, Delete). En nuestro caso únicamente se realizan peticiones Read.

El DAO es el encargado de establecer la conexión con la base de datos y realizar las operaciones que desea el cliente. De este modo, el cliente está totalmente desacoplado al acceso a datos, lo que significa que no importa la manera ni dónde se almacenen los datos, no afecta a la lógica del cliente. En este caso, el DAO crea un objeto DTO con los datos obtenidos de la consulta. Un DTO es una objeto plano que recoge una serie de atributos y métodos de acceso a dichos atributos que tiene como finalidad transferir la información desde la capa de servicios a la capa del cliente.

Para reducir la complejidad de la aplicación, se decide realizar crear una clase abstracta Spot que sirva de padre para los diferentes tipos de puntos que haya dentro del juego (*AttackSpot* y *MovementSpot*) [Apéndice IV]. Esta clase contiene las funcionalidades comunes de todos los puntos, como es la de acceso a los datos cuando comienza el juego. Dentro del juego habrá determinadas localizaciones que serán puntos (*MovementSpot* o *AttackSpot*) los cuales formarán anillos concéntricos alrededor y dentro del castillo. Cada anillo estará numerado con una letra y un número del 1 al número de puntos de ese anillo.

En esta primera fase del proyecto no se dispone del escenario definitivo, por lo que se crea uno que servirá para ir realizando pruebas. Por otro lado, al no haberse implementado los controles para las gafas de realidad virtual, estas pruebas se realizarán haciendo que la interacción del usuario con el juego sea con el ratón del ordenador.

- **Pruebas**

Prueba 1	
Título	Guardar datos
Comportamiento esperado	Se guardan los datos de puntos de ejemplo según aparecen en el editor de Unity
Resultado de la prueba	OK

Prueba 2	
Título	Recuperar Datos
Comportamiento esperado	Se recuperan los datos de la base de datos online y estos aparecen en los puntos de ejemplo del juego
Resultado de la prueba	OK

Sprint 2

- **Requisitos**

  Historia 04

- **Tareas**

  - Realizar el diseño del cambio de estado de atacar a moverse y viceversa de la unidad atacante.
  - Diseñar el movimiento hacia delante y horizontal de fuera de la primera muralla.
  - Diseñar el comportamiento de la unidad para subir la muralla
  - Diseñar el camino que debe seguir la unidad atacante para bajar de las almenas a dentro de la primera muralla.
  - Diseñar el movimiento horizontal de dentro de la primera muralla.

- **Diseño**

  Para determinar el comportamiento del soldado atacante se crea una clase *EnemyAI* [Apéndice II] que contiene un enumerado que determina el comportamiento del personaje. Se añaden los estados MOVE, ATTACK y CLIMB al enumerado. Cuando el personaje llega a un punto del escenario, comprueba primero si puede atacar, luego si puede escalar, y por último mira hacia donde puede moverse. Si no puede moverse a ningún punto, permanece quieto. El punto en el que se encuentra es el encargado de determinar hacia dónde puede moverse.

- **Implementación**

  Para programar la lógica que sigue el atacante para moverse hasta el castillo se crea un script que irá conectado a una componente del personaje y que se encarga de ordenar que ataque o se mueva, cuando debe hacerlo y cómo debe hacerlo.

  Esta clase contiene además el punto sobre el que se encuentra el personaje y también el punto hacia el que se va a dirigir a continuación, si es que se está moviendo. En el método nativo Update de Unity se comprueba contínuamente el estado en el que se encuentra y se ejecuta uno u otro método dependiendo de este. Cuando el personaje está en el estado MOVE se comprueba primero si tiene un punto como destino al que moverse y si no es así, se escoge uno. El punto en el que se encuentra el personaje fuera de la muralla tiene los siguientes datos: puntos hacia los que moverse para avanzar (si se puede), puntos a los que moverse de manera lateral, puntos a los que se puede atacar desde esta posición y su distancia, si este punto es un punto donde aparecen los atacantes, el muro que se puede escalar sobre este punto y si desde este punto se puede entrar por la puerta al castillo o al patio interno.

  Cuando el agente llega al punto que tenía asignado como destino, pasa al estado de ATTACK para comprobar si puede atacar a alguien. Cuando está en estado de ataque, se recoge en dos listas los puntos y las murallas a las que puede atacar que estén dentro de su rango de ataque a partir de los todos los puntos a los que se puede atacar desde el punto actual, información ya calculada previamente. En siguiente lugar se crean dos listas, una con los puntos dentro del rango de ataque que contengan soldados y otra con los puntos dentro del rango de ataque que contengan unidades de artillería.

Para continuar, y al no haber muchas clases de unidades, se aborda con un switch el comportamiento que debe tener el soldado de acuerdo a su clase. Básicamente se comprueba si las listas con unidades a las que atacar y la lista con los muros están o no vacías. Si alguna no lo está, se escoge un punto aleatorio dentro de ellas y se dispara a la unidad que se encuentra en él.

El soldado entra en el estado CLIMB justo después de salir del estado ATTACK. Cuando está en CLIMB, comprueba que se encuentra en un punto que tiene asociada un muro y que dicho muro ha sido destruido. Para saber la vida de la muralla, se declara una clase aparte *Life* [Apéndice II] que guarda la vida inicial del objeto al que esté asignada y la vida actual, que puede actualizarse.

Cuando la vida llega a 0 se realiza una invocación de un evento de Unity, que sirve para invocar algún método en otra clase, siempre y cuando esta otra clase tenga una referencia al objeto *Life* y un *Listener* añadido a esta referencia. En este caso en concreto la clase puerta tiene un *Listener* al objeto *Life* de la propia puerta del juego y cuando este evento se desencadena, se invoca el método que elimina la maya de la puerta. La clase *Life* se utilizará también para los soldados.

Cuando se den estas condiciones, el personaje comprobará si hay o no una torre de asedio construida para subir el muro. Si no la hay, permanecerá quieto en el punto en el que se encuentra y la torre comenzará a construirse poco a poco delante de la muralla. Si ya está construida, avanza hacia la base de la torre y tras unos segundos, aparece en la parte superior de la torre. Para que esto suceda se crean dos puntos a mayores con sus respectivas clases: uno en la base (*SiegeTowerPoint*) y otro en la cima (*SiegeTowerDestinationPoint*) [Apéndice II]

- **Pruebas**

Prueba 1	
Título	Movimiento hacia delante
Comportamiento esperado	El personaje se mueve hacia uno de los puntos que tiene delante suyo siempre y cuando estos no estén ocupados y no tenga ningún enemigo en el rango de alcance y no pueda escalar.
Resultado de la prueba	FALLO: El personaje se mueve hacia puntos hacia los que hay otros personajes moviéndose, por lo que chocan y a veces no llegan al punto hacia el que se dirigían, por lo que no continúan moviéndose.

Prueba 2	
Título	Movimiento horizontal
Comportamiento esperado	El personaje se mueve hacia la izquierda o hacia la derecha de manera repetida hasta que pueda avanzar hacia algún punto de delante
Resultado de la prueba	FALLO: El personaje se mueve hacia izquierda o derecha de manera aleatoria. No sigue una única dirección. FALLO: Cuando se encuentra cerca de la muralla, los personajes ignoran a aquellos que están esperando a que se construya la

	torre de asedio e intentan ir hacia el punto en el que se encuentran.

Prueba 3	
Título	Atacar defensor
Comportamiento esperado	El personaje detecta que hay enemigos dentro del rango y por tanto, no se mueve del punto en el que está mientras ataca
Resultado de la prueba	OK

Prueba 4	
Título	Construir torre
Comportamiento esperado	El personaje detecta correctamente cuando está cerca de la muralla que una parte de la misma ha sido destruida y se queda quieto esperando un punto próximo a que la torre se construya.
Resultado de la prueba	OK

Prueba 5	
Título	Avanzar hacia la torre
Comportamiento esperado	El personaje avanza hacia la torre cuando llega a un punto cerca de un muro destruido y la torre de asedio está construida, siempre y cuando en la base de la torre de asedio no haya nadie
Resultado de la prueba	FALLO: No se detecta cuando otro personaje está avanzando hacia la base de la torre, por lo que hay choques.

Prueba 6	
Título	Subir la torre
Comportamiento esperado	Tras llegar a la base de la torre, pasados unos segundos el personaje aparece en la parte superior.
Resultado de la prueba	OK

Sprint 3

- **Requisitos**

  Completar Historia 04 e Historia 06.

- **Tareas**
  - Diseñar el movimiento hacia delante y horizontal de fuera de la primera muralla (Completar)
  - Diseñar el comportamiento de la unidad para subir la muralla (Completar)
  - Diseñar el camino que debe seguir la unidad atacante para bajar de las almenas a dentro de la primera muralla. (Completar)
  - Diseñar el movimiento horizontal de dentro de la primera muralla. (Completar)
  - Establecer un sistema de guardado de distancias entre cada punto de ataque a puntos clave del mapa, para establecer un mapa de distancias.
  - Programar el guardado de estas distancias clave en el script de guardado de datos.
  - Dividir los puntos de ataque del castillo de acuerdo al tipo de unidad que puede haber en ellos
  - Programar el camino de puntos más pequeño entre dos puntos de ataque teniendo en cuenta las almenas y el cambio entre pisos
  - Mostrar con una línea el camino de puntos más corto que se debe recorrer para llegar de un punto a otro en tiempo real.

- **Diseño**

  Se decide separar la parte en la que el soldado se queda quieto mientras la torre se construye de la parte en la que escala por ella. Se crea otro estado llamado BUILD para hacer referencia a lo primero.

  Para detectar con exactitud si un punto está o no siendo ocupado, es preciso señalar esa ocupación en el momento en el que un personaje sabe hacia qué punto se mueve. De este modo, cuando un personaje deja un punto, automáticamente pasa a estar ocupando el siguiente punto y entonces los otros personajes no pueden avanzar hacia este.

  Se crea una clase *Door* que haga referencia a las puertas del castillo y que tenga un comportamiento similar a los muros, teniendo una vida y pudiendo ser destruidas.

  Para la historia 6 se decide crear un sistema de guardado de distancias entre puntos. Para no saturar la base de datos y porque además sería muy ineficiente, se descarta guardar la distancia de cada punto a todos los puntos del propio castillo.

  Se decide que determinados puntos dentro del castillo estarán marcados como puntos clave. Se guardará entonces la distancia desde cada punto del castillo a su punto clave más cercano por la izquierda y por la derecha (ya que el movimiento dentro del castillo siempre es lateral, a menos que se baje o suba de una almena). La distancia calculada es una recta entre los dos puntos que ignora cualquier objeto que haya entre medias, por lo que puede no ser una representación realista de la distancia que el personaje debe recorrer entre un punto y otro. Por ello se distribuyen estos puntos de manera que esto no suceda.

  Para controlar el comportamiento de los defensores, se crea una clase *AllyAI* [Apéndice II] que tiene una estructura similar a *EnemyAI*. Como en este caso el defensor solo puede atacar o moverse, se decide no hacer un enumerado que represente una máquina de estados.

- **Implementación**

  En los puntos de movimiento se añade un atributo booleano que indica si ese punto está ocupado o no y también un atributo de tipo *Character* [Apéndice II] que hace referencia al personaje que está dirigiéndose hacia ese punto.

  De este modo ningún soldado puede avanzar hacia este punto si el booleano indica que está ocupado, solo aquel personaje cuya componente *Character* sea igual al atributo de tipo *Character* del punto en cuestión.

  Al igual que para saber a través de qué puntos se puede escalar qué muros, también se indica desde qué puntos se puede entrar por la puerta. Cuando un personaje llegue a un punto cerca de la puerta comprobará si está o no destruida y si lo está, entrará por ella yendo de punto en punto hasta penetrar en el castillo.

  Para determinar hacia dónde se dirige el atacante una vez suba la torre de asedio, se introduce en el punto de la cima de la torre de asedio *SiegeTowerDestinationPoint* [Apéndice II] una lista con puntos dentro de la muralla hacia dónde se debe mover y lo hace de manera aleatoria. Una vez se mueve a uno de estos, se desplaza hacia la izquierda o hacia la derecha hasta que se tope con una escalera.

  Para interrumpir el movimiento lateral y hacer que baje, se establece un punto de conexión *JumpPoint* [Apéndice II] y se hace que el atacante siempre se dirija al *JumpPoint* del punto actual si es que tiene uno.

  Para las puertas se utiliza el mismo sistema que los muros, con una componente *Life* y una maya de renderizado.

  Para guardar las distancias entre un punto cualquiera y sus puntos clave derecha e izquierda se utiliza el mismo DAO que para los *MovementSpot*, solo que un nuevo método. También se crea un nuevo DTO que recoge los datos.

Una vez guardados y recuperados estos datos, se calcula el camino de puntos más corto de un punto a otro en la clase *AllyAI*:

1. Primero se calcula el camino de puntos **clave** más corto desde un punto a otro y también se suma las distancias entre los puntos de ese camino, que ya están previamente calculadas.

    - **Camino entre puntos del mismo piso**

      Primero de todo se comprueba si los puntos están en el mismo piso. Para ello se dividen los puntos del primer piso y del segundo piso en dos capas y simplemente se comprueba a qué capa pertenece el punto desde el cuál se va a mover el defensor y el punto destino.

```
//Si el origen y el destino están en el mismo piso
if(CheckSameFloor(destination))
{
    //Y es el segundo piso
    if (_currentSpot.gameObject.layer == LayerMask.NameToLayer("AttackSpots2")){...}
    //Y es el primer piso
    else{...}
}
```

Figura 39: Comprobación puntos mismo piso

Si se verifica que están en el mismo piso, se distingue el caso en el que estén en el segundo piso o en el primero.

En medio del anillo que forman los puntos en el **segundo piso** se encuentra la torre del homenaje. Los puntos están numerados de manera que el punto 1 queda a la derecha de la torre y el punto más alto (50) queda justo a la izquierda. En el resto de anillo, un personaje podría moverse del punto 50 al 1 y viceversa, pero en este caso no, por estar la torre del homenaje en medio.

Debido a esto, si un personaje quiere moverse del punto 5 al 45, debe moverse primero al 6, luego al 7… hasta llegar al 45. **No puede ir por la otra dirección.**

En el caso en el que los puntos estén en el **primer piso** se hacen dos distinciones principales como se muestra en la Figura xxx. Si el punto origen y el punto destino tienen los mismos puntos clave a izquierda y a derecha, entonces es que están muy cerca uno del otro.

```
//Y es el primer piso
else
{
    //Si origen y destino tienen los mismos puntos clave o el destino es un punto clave del origen
    //o el origen es un punto clave del destino (Casos especiales para que funcione)
    if (SameKeySpots(currentAttackSpot, destination) == 2 ||
        KeySpotFromEachOther(currentAttackSpot, destination)
        || _currentSpot.tag == "C1"){...}

    //Si origen y destino tienen en común 1 solo punto clave, están cerca y se tarda menos
    //yendo por las almenas.
    else if (SameKeySpots(currentAttackSpot, destination) == 1){...}

    //Si la distancia entre origen y destino es muy grande, tardará menos bajándose de las almenas y yendo
    //por el suelo del primer piso.
    //Se comprueba si tarda menos por la derecha o por la izquierda.
    else{...}
}
```

Figura 40: Comprobación de la cercanía entre el origen y el destino en el primer piso

Si el punto origen y destino están muy lejos uno del otro, entonces el personaje tardará menos si baja de las almenas, recorre el primer piso y sube las escaleras más cercanas al punto destino que se encuentra en las almenas. Debido a que no todos los puntos clave de cada punto son exactamente puntos que estén en las escaleras, se decide añadir un nuevo atributo a los puntos que sea *closestStairs*, una lista con los puntos más cercanos que estén al lado de escaleras (aquellos

que tengan un jumpPoint). De igual modo que en el anterior caso, es necesario comprobar si se tarda menos por las escaleras de la derecha o de la izquierda. Para calcular este camino se divide en las siguientes partes:

1. **Calcular camino hasta escalera:**

   Se comprueba cada punto en una dirección hasta dar con uno que tenga un JumpPoint [Apéndice IV], ya que será una escalera. Se añaden a una lista los puntos clave y se suma la distancia.

2. **Calcular el camino hasta la escalera más cercana al punto destino.**

   En la misma dirección (izquierda o derecha) se comprueba punto a punto si tiene o no un JumpPoint que sea el mismo del punto clave del punto destino. Se añade a la lista los puntos clave y la distancia.

3. **Calcular el camino desde la escalera hasta el punto destino**

   Se añaden a la lista los puntos clave desde el punto en la escalera al destino y se suma la distancia.

- **Camino entre puntos de distinto piso**

Para subir y bajar de piso el procedimiento es bastante distinto.

En el caso de subir, primero el personaje debe calcular el camino más corto hasta la puerta que da al patio de armas. Para llevar a cabo este cálculo es necesario realizar dos caminos, uno que tome las escaleras de bajada más cercanas por la izquierda y otro que tome las escaleras de bajada por la derecha.

Se calculan dos entonces dos caminos, uno que lleve desde el punto inicial a la escalera derecha más cercana y otro a la escalera izquierda más cercana. Una vez se baja por las escaleras, a partir de ese punto se realizan otros dos caminos, uno a la derecha y otro a la izquierda del anillo, buscando los puntos marcados como clave y sumando las distancias. El objetivo es llegar a uno de los puntos clave del punto destino, **tomándose en este caso el punto de la puerta como tal.** Una vez se llega a uno de estos puntos clave, se añade el punto de la puerta y se termina.

Para simular que el personaje sube por alguna escalera dentro del castillo y que termina en una de las salidas que da a las almenas de la segunda muralla, se establecen dos puntos dentro del patio de armas que se sitúan justo en las puertas que daban a alguna de estas escaleras.

Para saber a cuál de estos dos puntos ha de moverse la unidad y por cuál debería aparecer en el piso superior (ya que hay 3 salidas) se crea un método que calcula esta parte del camino de acuerdo al destino que tiene el personaje. Los puntos clave de los puntos de movimiento del piso superior son puntos que se encuentran exactamente al lado de las escaleras para moverse entre pisos, por lo

que simplemente se mirá el punto clave más cercano del punto destino y se designa ese como punto de salida al segundo piso. Además estos puntos al lado de la escalera están asociados a los puntos del comienzo de la escalera, por lo que también se añaden estos puntos.

Una vez en el piso de arriba, se sigue la misma metodología que si el personaje se moviera entre dos puntos de ese piso.

Para bajar de piso el procedimiento es bastante similar aunque con pequeños matices.

Obviamente, el cálculo del camino de puntos clave comienza al revés. Primero el camino entre el origen y la escalera para bajar más cercana, los puntos por los que baja y la puerta del patio.

A continuación se calculan dos caminos, uno a la izquierda y otro a la derecha, teniendo como origen el punto más cercano a la puerta. Cada uno de estos caminos continúa comprobando y añadiendo a la lista los puntos clave en la misma dirección. Cuando uno de los puntos es la bajada de una escalera cercana al punto destino, entonces se añade el punto de la escalera y, teniendo que este punto escalera y el punto origen forman parte del mismo anillo, se calcula si el camino más corto es por la izquierda o por la derecha y una vez hallado, se añaden los puntos clave en medio hasta el destino.

Como en cada cálculo que se hace del camino más corto se va sumando la distancia entre puntos clave, al final se obtiene no solo un camino de puntos clave desde el origen al destino sino también la distancia total de recorrido. Al final se comprueba la distancia menor y se escoge como camino más corto el que tenga esa distancia mínima.

2. En segundo lugar, se calcula el camino de puntos exacto a partir del camino de puntos clave. Para calcular esto se deben tener en cuenta una serie de características.

   A pesar de que la numeración de los puntos de un anillo está hecha de manera ascendente hacia la derecha, no siempre que el número del punto clave actual en la lista de puntos clave sea menor que el siguiente quiere decir que entonces haya que añadir los puntos que se encuentran en el medio de ambos (lo que sería avanzar hacia la derecha).

   Por ejemplo: si el número del punto actual es 4 y el siguiente es 90 es probable que los puntos que haya en el medio de ambos y que el personaje debería seguir sean el 3, 2, 1… 92, 91, 90. Del mismo modo sucedería si se invirtiera el orden.

   Para determinar con exactitud esto se usa el algoritmo mostrado en la Figura 41.

```
if(numeroOrigen > numeroDestino)

    if(numeroOrigen - numeroDestino >= (numeroPuntosEnElAnillo -
    numeroOrigen +   numeroDestino)

        Moverse a la derecha

    else

        Moverse a la izquierda

else

    if(numeroDestino - numeroOrigen >= (numeroPuntosEnElAnillo -
    numeroDestino+     numeroOrigen )

        Moverse a la izquierda
    else

        Moverse a la derecha
```

Figura 41: Pseudocódigo del algoritmo para decidir si el personaje se mueve a derecha o izquierda

Básicamente se compara cuántos números hay entre el número origen y el número destino y cuántos números hay entre el número origen y el número de puntos o entre el número origen y el 1, dependiendo de si el origen es más grande que el destino, y entre el número de puntos y el destino o entre el número 1 y el destino. Si hay menos números de una forma que de otra, entonces los puntos con esos números son los añadidos a la lista real.

Si se encuentra un punto de escaleras, se añaden los puntos de la lista de puntos clave hasta que se encuentre un punto de movimiento, en cuyo caso se siguen añadiendo los puntos de manera normal.

Cuando se está calculando los puntos del piso superior, simplemente se añaden los puntos que hay en el medio del punto origen y el destino, ya que los puntos de movimiento del piso superior no forman un anillo exacto al estar cortado por la torre del homenaje, y entonces el camino no puede tener la secuencia de puntos 1 – número del punto más grande o número del punto más grande – 1.

Una vez se tiene calculado el camino de puntos real, se tiene que programar algo que cree una línea visual que una estos puntos para indicar al jugador por dónde se moverá el personaje en cada caso. Esta línea además debe actualizarse cuando el jugador elige otro destino posible, junto a los puntos que forman el nuevo camino más corto.

Para conseguir trazar una línea visual entre el origen y el destino se utiliza el componente *Line Renderer* de Unity, que dado un array que contenga dos o más posiciones en unas coordenadas 3D, dibuja una línea recta entre cada posición. Para controlar cuando esta línea debe o no aparecer se hace lo siguiente: con el método nativo de Unity *OnDragMouse()* se registra cuando el usuario hace click sobre una unidad aliada siempre y cuando esté en la fase de batalla. Por otro lado,cada punto de ataque utiliza la función *OnMouseOver()* para detectar cuando el usuario pasa el ratón por encima suyo. Si mientras lo hace, se detecta que hay una

unidad seleccionada y el punto sobre el que está el ratón es un punto en el cuál puede situarse la unidad de acuerdo a su clase, entonces se marca ese punto como destino del camino.

Cuando el personaje detecta que hay un punto de ataque marcado como destino, entonces es cuando se realizan los cálculos del camino más corto y se dibuja la línea. Si el usuario deja de mantener presionado el botón, evento que se detecta con *OnMouseUp()*, y hay un punto de ataque seleccionado, entonces el soldado comenzará a recorrer el camino punto por punto hasta el destino. En caso contrario, entonces se desmarca el personaje actual como seleccionado y el dibujo del camino desaparece.

El paso siguiente es construir una maya de navegación para los agentes del juego. Esto se hace en la ventana *Navigation* proporcionada por Unity. Para hacer esta maya de navegación es necesario proporcionar unos parámetros del agente que se va a mover por el terreno. Una vez proporcionados unos parámetros más o menos precisos del personaje general del juego se procede a calcular la maya.

- **Pruebas**

Prueba 1 (De nuevo)	
Título	Movimiento hacia delante
Comportamiento esperado	El personaje se mueve hacia uno de los puntos que tiene delante suyo siempre y cuando estos no estén ocupados y no tenga ningún enemigo en el rango de alcance y no pueda escalar.
Resultado de la prueba	OK

Prueba 2 (De nuevo)	
Título	Movimiento horizontal
Comportamiento esperado	El personaje se mueve hacia la izquierda o hacia la derecha de manera repetida hasta que pueda avanzar hacia algún punto de delante
Resultado de la prueba	OK

Prueba 3 (De nuevo)	
Título	Avanzar hacia la torre
Comportamiento esperado	El personaje avanza hacia la torre cuando llega a un punto cerca de un muro destruido y la torre de asedio está construida, siempre y cuando en la base de la torre de asedio no haya nadie

Resultado de la prueba	OK

Prueba 4	
Título	Subir la torre
Comportamiento esperado	Tras llegar a la base de la torre, pasados unos segundos el personaje aparece en la parte superior.
Resultado de la prueba	OK

Prueba 5	
Título	Subir la torre
Comportamiento esperado	Tras llegar a la base de la torre, pasados unos segundos el personaje aparece en la parte superior.
Resultado de la prueba	OK

Prueba 6	
Título	Subir la torre
Comportamiento esperado	Tras llegar a la base de la torre, pasados unos segundos el personaje aparece en la parte superior.
Resultado de la prueba	OK

Prueba 7	
Título	Guardar datos
Comportamiento esperado	Se guardan correctamente en la base de datos las distancias desde cada punto a sus dos puntos clave y también se guarda si un punto es clave o no
Resultado de la prueba	OK

Prueba 8	
Título	Recuperar datos
Comportamiento esperado	Se recuperan los datos relativos a los puntos clave de manera correcta

Resultado de la prueba	OK

Prueba 9	
Título	Camino más corto entre puntos del mismo piso
Comportamiento esperado	Se calcula correctamente los puntos que forman el camino más corto entre puntos del mismo piso
Resultado de la prueba	FALLO: Si el origen o el destino es un punto clave, el camino se calcula de manera incorrecta. FALLO: El personaje no puede moverse a puntos que se encuentren en una torre.

Prueba 10	
Título	Camino más corto para subir al segundo piso
Comportamiento esperado	Se calcula correctamente los puntos que forman el camino más corto desde un punto del primer piso a un punto del segundo piso
Resultado de la prueba	FALLO: La parte del camino que va desde una escalera a la puerta no siempre es es la más corta posible

Prueba 11	
Título	Camino más corto para bajar al primer piso
Comportamiento esperado	Se calcula correctamente los puntos que forman el camino más corto desde un punto del segundo piso a un punto del primer piso
Resultado de la prueba	FALLO: El camino de la puerta a las escaleras cercanas al punto destino no siempre se calcula correctamente.

Sprint 4

- **Requisitos**

  Corregir fallos de la historia 06 y avanzar historia 11 y 12

- **Tareas**

  - Programar el guardado de estas distancias clave en el script de guardado de datos (Completar)
  - Programar el camino de puntos más pequeño entre dos puntos de ataque teniendo en cuenta las almenas y el cambio entre pisos. (Completar)
  - Dividir los puntos de ataque del castillo de acuerdo al tipo de unidad que puede haber en ellos

- Establecer características que comparan todas las armas y asignar determinados valores a cada una según se recoja información de la documentación histórica
- Establecer una fórmula que calcule el daño causado por un arma teniendo en cuenta las características de esta y el objetivo al que dispara.
- Establecer un sistema de creación de objetos que simbolicen los proyectiles que disparan las armas y qué características tienen.
- Crear un gestor de partida que controle el transcurso del juego

- **Diseño**

Se crea un nuevo atributo en la clase *MovementSpot* [Apéndice IV] *towerSpot* que guarde un punto de la clase *TowerSpot* [Apéndice IV] y que simboliza el punto central situado en cada torre.

Para dividir los puntos del castillo de acuerdo al tipo de unidad que puede haber en ellos se crean diferentes capas para cada anillo, de manera que un tipo de unidad solo se pueda mover a los anillos que formen parte de una en concreto.

Para gestionar la partida se crea una clase *GameManager* [Apéndice II] que controle cuando el juego se encuentra en medio de una oleada y cuando no.

Para la lógica de los disparos es necesario trabajar con el motor de físicas de Unity. Se tienen que señalar las colisiones que debe haber entre cada proyectil y el resto de objetos y también qué sucede cuando dichas colisiones se dan. Además hay que determinar factores cómo cada cuanto tiempo se puede disparar cada proyectil y el daño que hace [Apéndice V].

- **Implementación**

Los errores que se tenían para las pruebas 9, 10 y 11 eran en su mayoría errores de código, no de diseño.

Para que el personaje se pueda mover a un punto situado en una torre en el cálculo final de puntos exactos que tienen que recorrer, se comprueba tanto al principio como al final si el origen o el destino están en una torre y si es así, se añade este punto a la lista.

Se crea una clase central del proyecto llamada *GameManager*, que se encargará de controlar el desarrollo del juego y ser el cerebro de la aplicación. Entre sus funciones están el pasar de la fase de colocación de tropas a la fase de batalla, controlar cuando aparecen los enemigos y cuándo y cómo el jugador coloca unidades, y guardar información general de la aplicación la cuál no sería correcto guardar en clases concretas. Además es el encargado de comprobar cuando el jugador gana o pierde y realizar las correspondientes funciones.

Como no se quiere que haya más de un objeto *GameManager* en la aplicación, se utiliza el patrón *Singleton*, que evita que se cree más de un objeto de una clase. Se provee también un punto de acceso global a través de instancia compartida estática.

Para que los disparos funcionen de acuerdo a la lógica del mundo real, se hacen dos cosas.

- ☐ Se crea un script que simplemente provoca que el objeto al que está añadido se mueva en la dirección x del vector de su origen a una velocidad ajustable desde el editor. De este modo cuando se instancie una proyectil en el videojuego, este tendrá una velocidad constante en la dirección x.

- ☐ Por otro lado, se crea otro script que registre cuando el proyectil entra en contacto con un objeto.

Si ese objeto tiene una componente Life, entonces la cantidad de vida actual de ese objeto se resta un valor igual al daño que tenga ese proyectil, el cuál se asigna nada más instanciarse este dependiendo de qué arma sea la que dispara.

- ☐ A mayores, si el proyectil no impacta con nada pasados unos segundos, entonces el objeto se destruye, para evitar que quede en escena indefinidamente.

Con esto se tiene ya un planteamiento concreto y lógico sobre cómo deben ser los disparos en el juego. Para establecer los objetos que pueden entrar en contacto con el proyectil se utiliza la maya de colisión de capas de Unity, que se encuentra en la ventana *Project Settings*, en el apartado *Physics*. En esta maya se puede seleccionar los objetos de qué capas pueden entrar en contacto con los objetos de qué otras capas.

Se declara una capa para los defensores y otra para sus proyectiles y además otra para los atacantes y otra sus proyectiles. De este modo se puede asignar con facilidad que los proyectiles de los defensores solo entren en contacto con los atacantes (evitando el fuego amigo) y que los proyectiles de los atacantes solo choquen con los defensores. Además se añade que estos últimos proyectiles contactan con los muros y las puertas.

No obstante, buscando información sobre la mejor manera de gestionar proyectiles en videojuegos, se descubre que en la mayoría de los casos, es mejor utilizar el patrón de creación *Object Pooling*. [47]

Existen dos motivos principales para la justificación de este patrón. El primero es que las operaciones de instanciación y de destrucción de objetos son relativamente costosas si se comparan con el resto de operaciones de Unity y abusar de ellas puede suponer un impacto en la fluidez de la aplicación. El segundo es la fragmentación de memoria [48] que puede derivarse de esta creación y destrucción continuada, ya que la destrucción de un objeto libera el espacio en memoria que estaba usando, pero si el espacio liberado no es lo suficientemente grande como para que un objeto más grande entre, entonces ese espacio queda inservible.

El patrón de creación *Object Pooling* consiste en una clase única del proyecto que, cuando comienza la aplicación, crea, instancia una cantidad de objetos que determinada de un *prefab*, los desactiva y los introduce en una lista. Las clases que quieren instanciar un objeto de este prefab, lo que hacen es llamar a un método de la clase *Object Pooling* para que les devuelva el último objeto de la lista no activo y es responsabilidad de la clase activarlo cuando quiera.

De este modo se evitan las operaciones constantes de instanciación y de destrucción durante la partida. Solo se tiene una operación costosa al inicio en la que se crean estos objetos pero al estar inactivos desde su creación, impactan mucho menos en el rendimiento. Para aplicar el patrón de manera completa, es necesario que la parte del código en la que se destruía este objeto ahora creado por el *Object Pooling*, pase en cambio a desactivar el objeto.

- **Pruebas**

Prueba 1 (de nuevo)	
Título	Camino más corto entre puntos del mismo piso
Comportamiento esperado	Se calcula correctamente los puntos que forman el camino más corto entre puntos del mismo piso
Resultado de la prueba	OK

Prueba 2 (de nuevo)	
Título	Camino más corto para subir al segundo piso
Comportamiento esperado	Se calcula correctamente los puntos que forman el camino más corto desde un punto del primer piso a un punto del segundo piso
Resultado de la prueba	OK

Prueba 3 (de nuevo)	
Título	Camino más corto para bajar al primer piso
Comportamiento esperado	Se calcula correctamente los puntos que forman el camino más corto desde un punto del segundo piso a un punto del primer piso
Resultado de la prueba	OK

Prueba 4	
Título	Camino más corto para llegar a un punto en una torre.
Comportamiento esperado	Se calcula correctamente los puntos que forman el camino más corto desde un punto cualquiera a un punto de la torre
Resultado de la prueba	OK

Prueba 5	
Título	Camino más corto desde un punto de la torre
Comportamiento esperado	Se calcula correctamente los puntos que forman el camino más corto desde un punto de la torre a otro punto
Resultado de la prueba	OK

Prueba 6	
Título	Creación proyectiles
Comportamiento esperado	Los proyectiles se crean al inicio del juego y están desactivados.
Resultado de la prueba	OK

Prueba 7	
Título	Activación proyectil
Comportamiento esperado	El proyectil se activa en la escena cuando un personaje dispara.
Resultado de la prueba	OK

Prueba 8	
Título	Dirección proyectil
Comportamiento esperado	El proyectil se mueve siempre hacia delante desde el arma del personaje que dispara y en la dirección en la que se encuentra el objetivo
Resultado de la prueba	OK

Prueba 9	
Título	Colisión proyectil
Comportamiento esperado	El proyectil se desactiva cuando colisiona con algún elemento que pertenece a la capa con la cuál el proyectil puede colisionar y se desactiva.
Resultado de la prueba	OK

Prueba 10	
Título	Autodestrucción proyectil
Comportamiento esperado	Si el proyectil no colisiona con nada pasados unos segundos, este se desactiva de la escena
Resultado de la prueba	OK

Prueba 11	
Título	Gestor del juego
Comportamiento esperado	El gestor del juego *GameManager* solo tiene una única instancia. Los enemigos realizan acciones solo si se encuentran en medio de una oleada. El jugador tampoco puede mover sus unidades en este caso.

Resultado de la prueba	OK

Sprint 5

- **Requisitos**

  Historia 10

- **Tareas**

  - Crear un gestor de oleadas
  - Determinar los datos que debe guardar una oleada
  - Determinar los enemigos que tendrá cada oleada y la recompensa de cada una
  - Detectar cuando un enemigo muere y cuando el número de enemigos restantes de la oleada es 0.

- **Diseño**

  Para la gestión de oleadas se crea una clase *WaveManager* [Apéndice VI] que gestione la creación de cada oleada, la aparición de los enemigos de la misma y que controle cuando los enemigos de la oleada son todos destruidos.

  Se crea una clase *Wave* [Apéndice VI] que guarde los datos de cada oleada: el número de enemigos, la recompensa y además el número de enemigos restante que queda en la oleada.

  Para la instanciación de los enemigos se utilizan las corrutinas de Unity [ ].

- **Implementación**

  Al igual que para el GameManager, se utiliza el patrón Singleton para tener solo una instancia de la clase WaveManager. En esta clase se comprueba en el Update() cuando se acaba el periodo en el que el jugador puede colocar tropas y se comienza a instanciar enemigos.

  Primero se crea una oleada, en ella se guarda la recompensa de la oleada y el número de enemigos que tiene. A continuación se determina de qué tipo es cada enemigo de acuerdo al número de ronda que se esté desarrollando en ese momento. Por último se instancian los enemigos de uno en uno de manera aleatoria cada 0.5 segundos en un punto aleatorio del anillo de puntos más externo del escenario. Para realizar esta espera es necesario utilizar la operación WaitForSeconds(float seconds) de Unity. Esta operación sólo funciona si el método en el que se encuentra es de tipo IEnumerator y este método se invoca con la orden StartCoroutine().

  Cuando un atacante muere (Life indica que su vida llega a 0 en la Figura xxx) se desencadena un evento y se resta el número de enemigos que quedan en la oleada. Cuando el número de enemigos en la oleada llega a 0, se indica al GameManager que cambie la fase de la partida y a partir de entonces el jugador no puede reposicionar sus tropas pero puede comprar nuevas y también mejoras.

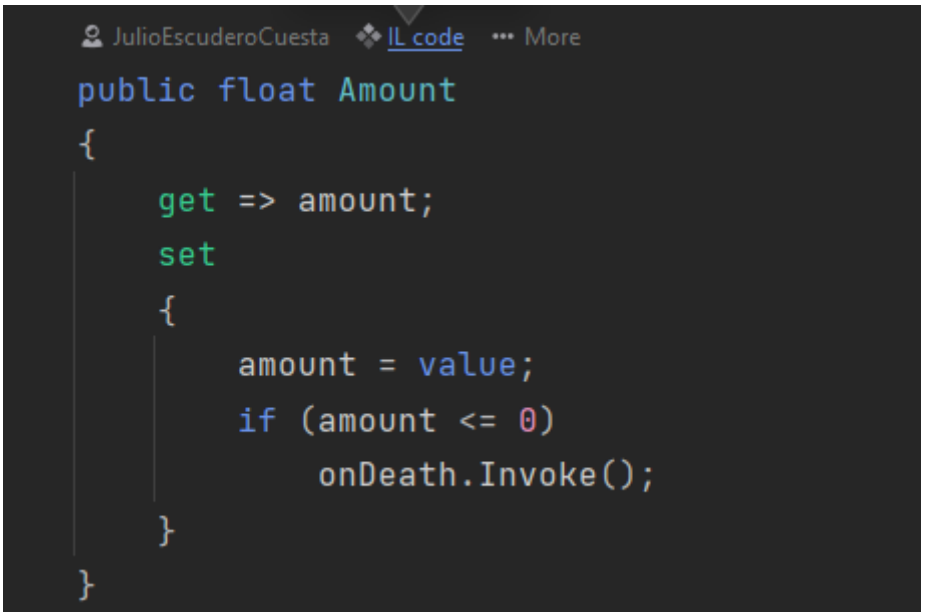


Figura 42: Invocación evento de Unity cuando la vida baja de 0

- **Pruebas**

Prueba 1	
Título	Gestor del oleadas
Comportamiento esperado	El gestor de oleadas solo
Resultado de la prueba	OK

Prueba 2	
Título	Creación oleada
Comportamiento esperado	La oleada se crea con el número de enemigos y recompensa planeados
Resultado de la prueba	OK

Prueba 3	
Título	Tipo de enemigos
Comportamiento esperado	El tipo de los enemigos se corresponde con el indicado de acuerdo al número de la ronda
Resultado de la prueba	OK

Prueba 4

Título	Detectar muerte de enemigos
Comportamiento esperado	Se detecta cuando muere un enemigo y cuando el número de enemigos restantes llega a 0, se cambia de fase, permitiendo al jugador comprar nuevas unidades y/o mejoras
Resultado de la prueba	FALLO: no se detecta de manera correcta cuando el enemigo muere ni cuando hay 0 enemigos en la oleada

Conclusiones

Tras lo expuesto, se puede concluir que han logrado los objetivos inicialmente propuestos. Se ha recogido la documentación histórica necesaria para generar la ambientación adecuada en el videojuego, se ha analizado las características principales de los juegos de esta índole y se ha diseñado y programado la mayor parte de la lógica del videojuego recogida en el Documento de Diseño del Videojuego [El proyecto ha servido para sentar las bases de un videojuego *Tower Defense* para las gafas de realidad virtual Oculus Quest 2 y sirve de punto de partida para seguir desarrollando el videojuego y conseguir una versión totalmente operativa y competente que pueda lanzarse al mercado.

La complejidad de creación de un juego como el que se ha planteado hace imposible que su implementación se pueda realizar en el tiempo que se tiene para la realización del TFG. Nunca se ha pretendido que esto fuera así. Sin embargo, el presente proyecto sí que ha servido para sentar unas bases sólidas para la creación de un videojuego Tower Defense para las gafas de realidad virtual Oculus Quest 2 y sirve de punto de partida para seguir desarrollando el videojuego y conseguir una versión totalmente operativa y probada que pueda lanzarse al mercado.

Como experiencia personal queda patente que en el sector del desarrollo de videojuegos la necesidad de disponer de un equipo multidisciplinar y con una formación cualificada y experiencia previa es casi fundamental para garantizar que el producto final tiene una calidad mínimamente aceptable para que un usuario pueda disfrutar de este.

Durante el desarrollo del proyecto se ha aprendido a programar en C#, se ha aprendido sobre el desarrollo de videojuegos, se han aprendido patrones de diseño nuevos, y también se ha aprendido sobre algoritmia, bases de datos no relacionales, además de el uso de Unity.

Trabajos Futuros

Cómo línea de trabajo futuro se espera poder terminar el videojuego y que cumpla con los requisitos recogidos al inicio del proyecto.

Más adelante, se pretende incluir nuevas características a las unidades y a las posiciones de defensa para que el cálculo del daño que realice una unidad no sea tan lineal, sino que haya más factores a tener en cuenta, como la defensa extra que puedan proporcionar determinadas posiciones del castillo o la armadura equipada que tengan las unidades. Además, se pretende ampliar el sistema de mejoras para que el jugador pueda tomar aún más estrategias.

También se planea que la creación de nuevas oleadas durante la partida tenga en cuenta datos de las oleadas anteriores para incluir un tipo de unidad u otro para dificultar dicha oleada de acuerdo al nivel en el que el jugador haya seleccionado jugar la partida.

Por otro lado, en futuras versiones se quiere también que haya un marcador en línea dónde los jugadores puedan ver la puntuación de otros jugadores e incluso entrar más en detalle en sus partidas para ver nuevas tácticas de defensa. También se pretende establecer un modo de juego multijugador en el que dos jugadores se enfrenten, uno siendo el atacante y otro el defensor.

Apéndices

Apéndice I: Documento de Diseño del Juego (GDD)

DOCUMENTO DE DISEÑO DEL VIDEOJUEGO

IDEA DEL JUEGO

INTRODUCCIÓN

La perspectiva del videojuego será top-down para las gafas de realidad virtual Oculus Quest 2. El jugador debe defender el Castillo de la Mota de Medina del Campo de enemigos invasores que atacan en oleadas.

GÉNERO

El videojuego recoge características del género *Tower Defense* aunque los enemigos no recorren un camino predefinido en un mapa alrededor del cuál el jugador coloca construcciones y cuarteles con soldados que atacan a estos enemigos y evitan que llegue a una posición del mismo. La única construcción en el juego es el castillo y la meta que en este caso los enemigos deben destruir es la torre del homenaje que se encuentra rodeada por las murallas del castillo y el jugador solo puede comprar diferentes tipos de unidades defensoras.

Además, el juego contiene elementos educativos, ya que todos los soldados y armas están creadas a partir de información recogida en la documentación histórica de la época.

PLATAFORMA

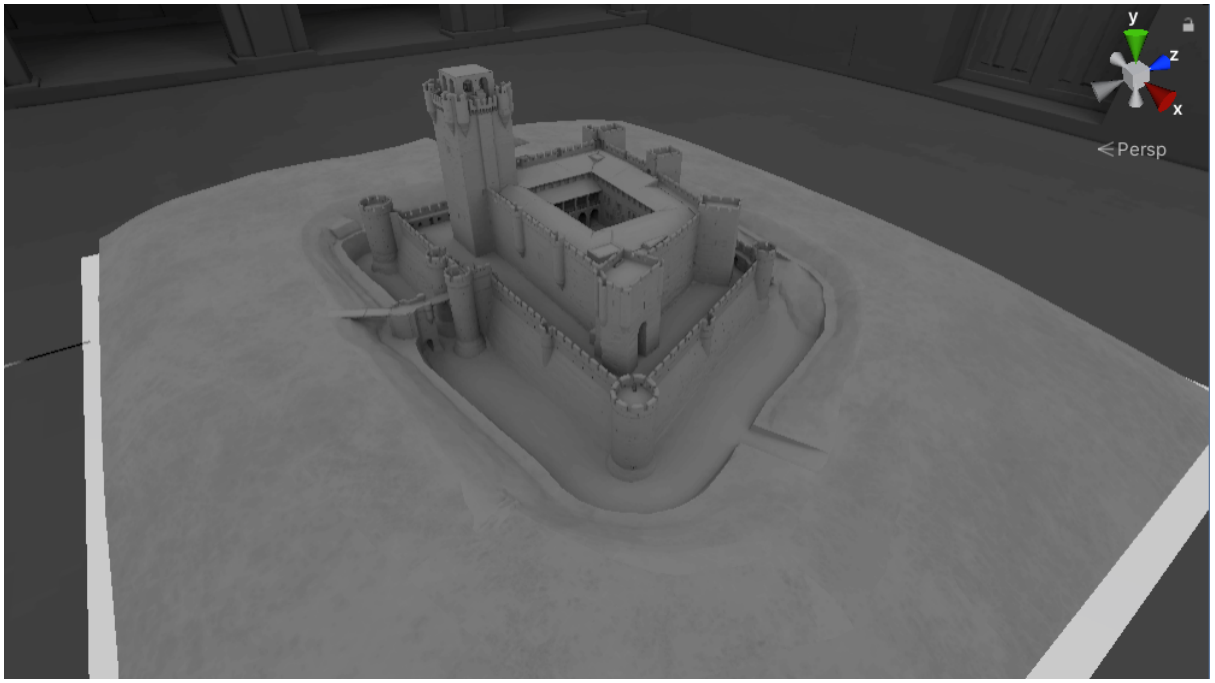
La plataforma objetivo del desarrollo son las gafas de realidad virtual Oculus Quest 2, aunque no se descarta la portabilidad del juego a otras plataformas como Playstation o Xbox.

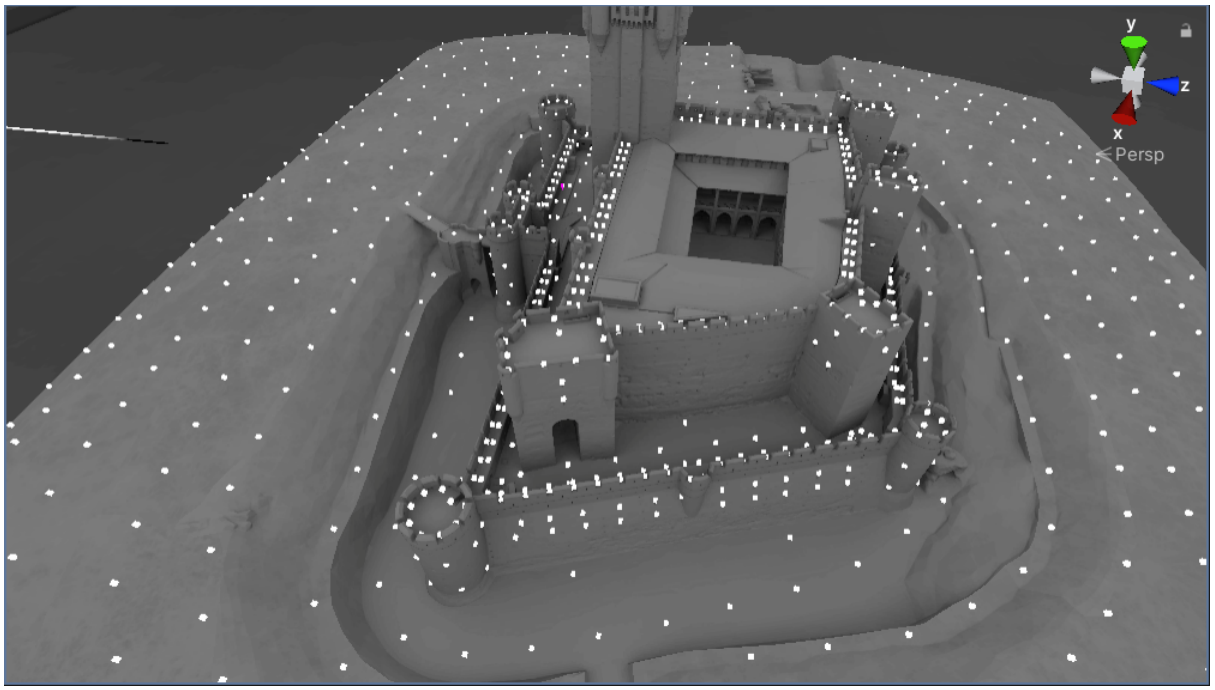
ESCENARIO

TERRENO DE JUEGO

La partida se desarrolla en una maqueta con una llanura en cuyo centro se encuentra el Castillo de la Mota. El jugador puede moverse alrededor de la maqueta para visualizar el terreno desde distintas posiciones y además también puede alejarse y acercarse para observar con más detalle.

Las unidades, tanto aliadas como enemigas, se mueven por el mapa desplazándose entre determinados puntos previamente definidos. Estos puntos forman anillos concéntricos desde el exterior al interior cada vez más pequeños y dentro de castillo se sitúan además en los diferentes puestos desde los que se atacaban en la época en la que está ambientado el juego.




y
z
x
Persp

MECÁNICAS DEL JUEGO

FLUJO DE EVENTOS

El desarrollo de la partida es bastante lineal. El jugador comienza en una fase de colocación en la que puede colocar soldados en las posiciones de defensa del castillo y comprar mejoras, todo ello a cambio de dinero.

Cuando el jugador quiera colocar un tipo de unidad en el mapa, seleccionará el tipo de unidad en el menú de tropas y a continuación podrá colocar esa unidad tantas veces como desee, siempre y cuando tenga el dinero disponible. Además, en el mapa aparecerán resaltados los puntos en los que puede colocar esa unidad y el jugador no podrá colocar esa unidad en otros lugares. Cuando el jugador tenga el puntero encima de un punto, se mostrará las características extra que pueda ganar el soldado si se posiciona en ese punto. El dinero de la compra se restará del dinero total del jugador únicamente cuando seleccione un punto válido para el tipo de unidad que quiere colocar.

Para comprar las mejoras, el jugador simplemente tendrá que seleccionar la mejora deseada y se realizará la compra siempre y cuando tenga dinero suficiente.

Las opciones de compra de unidades y mejoras estarán sujetas a las compras previamente realizadas.

Cuando el jugador termina de organizar la defensa para la siguiente oleada, selecciona en el menú principal la opción *Siguiente Oleada* y tras unos breves segundos, comienzan a aparecer los enemigos de la oleada.

A partir de este momento el jugador no podrá adquirir ninguna unidad o mejora extra. Solo podrá mover unidades de un punto a otro (siempre y cuando el tipo de punto coincida con el tipo de unidad). Las unidades que mueva no podrán volver a ser movidas hasta que hayan llegado al punto designado como destino.

Cuando se eliminan todos los enemigos de una oleada, se recibe la recompensa propia y el jugador vuelve a poder comprar unidades y a adquirir mejoras.

Esta secuencia se repite hasta que el jugador logra superar la última ronda, en cuyo caso gana la partida, o hasta que los atacantes consigan destruir la torre del homenaje (su vida llegue a 0), en cuyo caso es derrotado.

MOVIMIENTO SOLDADOS

Como ya se ha explicado en el primer apartado, las unidades, tanto defensivas como atacantes se mueven por determinadas posiciones del mapa.

Los atacantes aparecen hasta los puntos del anillo más exterior de manera totalmente aleatoria. Priorizan moverse hacia alguno de los 3 puntos más cercanos del siguiente anillo y si no, se mueven hacia la izquierda o la derecha.

Las unidades de artillería avanzan hasta una distancia media entre el anillo más externo y el castillo y ya no avanzan más hacia delante, solo se mueven de manera lateral. Las unidades de infantería y los arqueros pueden seguir avanzando hasta llegar al foso del castillo o hasta que estén cerca de la puerta exterior y puedan entrar en el castillo por el puente levadizo. Si están en el foso y la muralla está parcialmente destruida, procederán a construir una torre de asedio que tomará 1 minuto y que les permitirá acceder a la primera muralla por las almenas.

Dentro del castillo las unidades solo se mueven de manera lateral y buscan escalar la segunda muralla construyendo más torres de asedio o acceder al patio a través de la puerta interior. Además pueden moverse a las torres si hay defensores apostados en ellas.

Las unidades del jugador solo se mueven de su posición si así lo ordena el propio jugador. Cuando se mueven, lo hacen por el camino más corto.

Tanto los atacantes como los defensores detienen su movimiento si detectan que hay una unidad a la que pueden disparar desde su posición y dentro de su rango de ataque.

SISTEMA DE OLEADAS

Las oleadas de enemigos tienen una dificultad incremental y están programadas haya variedad en las unidades que aparecen. Cuanto más avanza la partida, más fuertes serán estas unidades (adquirirán las mejoras propias), más unidades aparecerán y aparecerán con más frecuencia.

Determinados tipos de unidades, como el ariete o el trabuquete, aparecerán cuando se den ciertas condiciones en la partida. El ariete aparecerá en cada ronda a partir de que las torres con el puente levadizo sean destruidas y el trabuquete aparecerá cuando el propio jugador desbloquee la mejora trabuquete (de momento se toma esta decisión).

En principio, se ha decidido que cada oleada de una cantidad de recompensa determinada, aunque no se descarta establecer una fórmula que tenga en cuenta el tiempo empleado para derrotar a los enemigos y el daño recibido para variar la recompensa obtenida por el jugador. También se decide que el número inicial de enemigos sea de 10 y que aparezcan 10 enemigos más en cada oleada.

Para la proporción del tipo de los enemigos se decide que para las primeras oleadas 2 de cada 3 enemigos sean de infantería o arqueros y el resto sea artillería. A partir de la ronda 10 se introducirán la aparición de morteros enemigos en las oleadas.

Se decide que algunas oleadas tengan 1 mortero por cada 10 unidades y que haya menos unidades de otro tipo de artillería y que en otras oleadas haya menos unidades de infantería y arqueros pero más de artillería y morteros.

UNIDADES

Existen diferentes clases de unidades. Cada clase tiene características propias y contiene más de un arma con pequeñas diferencias unas de otras. Cada tipo de unidad puede conseguir una mejora que provoca que cambie de arma (dentro de la misma categoría) o que añada un accesorio al arma que ayuda a que elimine más unidades enemigas.

ARQUEROS

Los arqueros son unidades de corto daño y alcance, por lo que deben estar cerca de los enemigos para poder dispararlos. En contraposición, se pueden conseguir por un precio muy barato. Además, poseen una gran velocidad y cadencia para, en el caso de los atacantes, llegar rápido a la muralla y escalarla para disparar a los defensores y en el caso de estos últimos, para derrotar rápido a los atacantes antes de que estos pasen a disparar a los otros tipos de unidades

	Arco simple	Brazal	Arco compuesto	Dactilera	Flechas incendiarias	Ballesta	Gafa
1 Arqueros Unidades que disparan armas de arco y flecha. Ubicadas en las almenas. Prioridad de Objetivos: 1 Tropas - 2 artillería							
Longitud Total(cm)	120		120			90	
Estadísticas sobre 100 – Daño	1	1	3	3	5	10	10
Alcance	10	10	15	20	20	25	25
Cadencia	90	95	95	10	90	50	70
Punteria	15	15	15	15	15	20	20
Vida	30	30	30	30	30	30	30
Velocidad	100	100	100	100	100	100	100
Nº Soldados que operan	1	1	1	1	1	1	1
Modelo 3D	1_A	1_A	1_B	1_B	1_B	1_C	1_C
Modificaciones en textura y escala					La textura de las flechas debe ser diferente.		
Descripción	Arco con cuerpo formado por una única pieza.	Accesorio que reduce el esfuerzo al que se ve sometido la muñeca al apuntar y disparar que permite que el arquero aguante más tiempo disparando.	Arco con cuerpo formado por varias capas que le aportan una mayor resistencia y permiten un mayor tensado.	Protección que se usa en los dedos de la mano que tensa el arco para que la tensión soportada pueda ser mayor y así aumentar el alcance.	Práctica que consiste en prender fuego a la punta de la flecha con el fin de incendiar el objetivo disparado.	Arma que dispara flechas sin necesidad de mantener tensa la cuerda usando la fuerza de los brazos.	Elemento que permite un tensado del arma a la hora de recargar mucho más rápido.

INFANTERÍA

Las unidades de infantería son también unidades rápidas y con una gran cadencia. Son un poco más caras pero también tienen más daño y precisión, aunque pueden disparar menos veces por unidad de tiempo y necesitan estar prácticamente cuerpo a cuerpo para poder disparar.

2 Infantería

Armas portátiles de pólvota que pueden ser portadas por los soldados.

Prioridad de Objetivos: 1 Tropas - 2 artillería

	Trueno de mano	Cañón de mano	Espingarda	Cubre cazoletas	Arcabuz	Balas de plomo	Mosquete	Horquilla
Longitud Total(cm)	120	180	160		110		140	
Estadísticas Sobre 100 – Daño	5	7	10	10	13	16	20	20
Estadísticas Sobre 100 – Alcance	1	3	5	5	7	7	10	10
Estadísticas Sobre 100 – Cadencia	50	55	60	65	70	70	75	75
Estadísticas Sobre 100 – Punteria	50	55	60	60	65	65	80	100
Estadísticas Sobre 100 – Vida	30	30	30	30	30	30	30	30
Estadísticas Sobre 100 – Velocidad	100	100	100	100	100	100	100	100
Nº soldados que operan	1	1	1	1	1	1	1	1
Modelo 3D	2_A	2_B	2_C	2_C	2_D	2_D	2_E	2_E
Modificaciones en textura y escala							El arma consta de dos piezas. Debe ocultarse la horquilla.	La orquilla que estaba oculta se muestra.
Descripción	Arma de pólvora portátil primitiva que consta de un pequeño cañón que está unido a una estructura simple hecha de madera para ser sujetado.	Predecesor de los primeros fusiles que, como evolución de los primeros truenos de mano, evoluciona aumentando la longitud del cañón.	Fusil primitivo de avancarga de largo cañón difícil de disparar y, por lo tanto, poco preciso a la hora de apuntar.	Elemento que ayuda a prender la mecha frente a condiciones adversas como la lluvia o el viento.	Fusil de avancarga de menor longitud que su predecesor de corto alcance pero capaz de atravesar armaduras.	Evolución del material de los proyectiles.	Fusil de avancarga de mayor tamaño que el arcabuz, al que también superó en mayor precisión, alcance y potencia de fuego.	Accesorio que permite apoyar el arma al apuntar para hacerlo con una mayor precisión.

PIEZAS MENUDAS

Las unidades de artillería ligera se componen de pequeños cañones con bastante más daño que las unidades anteriores pero a costa de costar un poco más de dinero y a tener ratio de disparo menor, además de ser considerablemente más lentos. Por otro lado, pueden atacar a los muros del castillo, habilitando así a que los arqueros y la infantería pueda subir estos muros con las torres de asedio. En el lado del defensor, estas unidades solo se colocan en el subterráneo del castillo.

3 Piezas Menudas

Piezas alargadas de calibre pequeño.

Ubicadas en saeteras.

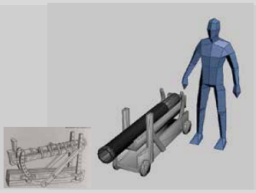
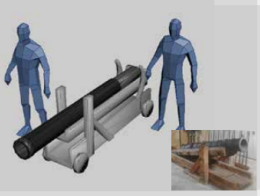
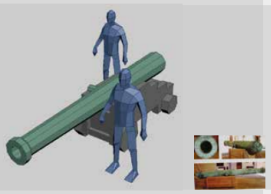
Prioridad de Objetivos: 1Cañones-2Personas-3muros

	Sacabuche	Medio Ribadoquín	Falconete	Sacre de Fonseca**	Media culebrina
Calibre (cm)	2,7	3,7	5,7	8,2	11,9
Longitud Total(cm)	140	180	168	350	415,7
Material	Bronce	Hierro	Hierro	Bronce	Bronce
Estadísticas Sobre 100 – Daño	25	30	40	50	60
Estadísticas Sobre 100 – Alcance	30	40	40	60	60
Estadísticas Sobre 100 – Cadencia	40	40	40	40	50
Estadísticas Sobre 100 – Punteria	50	60	60	50	50
Estadísticas Sobre 100 – Vida	30	30	30	40	50
Estadísticas Sobre 100 – Velocidad	60	60	60	50	50
Modelo 3D	3ABC (objetos 3ABC_B,C ocultos)	3ABC (objetos 3ABC_A,C ocultos)	3ABC (objetos 3ABC_A,B ocultos)	3DE (objetos 3DE_B ocultos)	3DE (objetos 3DE_,A ocultos)
Textura					
Nº Soldados que operan	1	1	1	1	2

PIEZAS GRUESAS

Las unidades de artillería pesada son similares a las anteriores pero tienen ciertas diferencias. La principal de ellas es la prioridad de ataque. Mientras que las piezas pequeñas sólo

atacarán a los muros cuando no tengan ningún objetivo al que atacar, las piezas gruesas priorizan los muros por encima de las tropas.
Además de esto, tienen un daño muy superior y más alcance, a costa de ser bastante más caras y moverse mucho más lento.

	Pasavolante	Bombarda	Culebrina De Juana I **	Cañón de fernando el católico
4 Piezas Gruesas Piezas de mayor calibre . Ubicadas en troneras. Objetivo: 1cañones-2Muros -3Personas .				
Calibre (cm)	15	26,5	13,6	18
Longitud Total(cm)	140	266	430	373
Material	Hierro Forjado	Hierro Forjado	Bronce	Bronce
Estadísticas Sobre 100 – Daño	70	90	80	100
Alcance	50	50	80	80
Cadencia	20	20	50	50
Punteria	50	40	60	60
Vida	40	40	80	100
Velocidad	40	20	40	30
Modelo 3D	4A	4B	4CD (cañón A)	4CD (Cañón B)
Textura				
Nº Soldados que operan	1	2	2	2

MORTEROS

Los morteros suponen una evolución a la artillería pesada. Aunque poseen prácticamente las mismas características, se distinguen en que su principal prioridad son los muros del castillo. Los morteros siempre buscarán muros a los que atacar por encima de todo. Además hay que destacar que **son unidades que solo utilizarán los atacantes. El jugador no tendrá acceso a ellos para defender el castillo**

	Cuartago	Trabuquera	Mortero
5 Morteros* Piezas de mayor calibre . *Solo utilizadas en Ataque. Objetivo: 1Muros-2Cañones -3Personas .			
Calibre (cm)	30	42	50
Longitud Total(cm)	65	76	90
Material	Hierro Forjado	Hierro Forjado	Hierro Forjado
Estadísticas Sobre 100 – Daño	70	90	100
Alcance	50	60	70
Cadencia	10	20	30
Punteria	10	20	30
Vida	40	40	50
Velocidad	20	20	20
Modelo 3D	5AB (cañón A)	5AB (Cañón B)	5C
Textura			Añadir Remates metálicos y oscurecer madera.
Nº Soldados que operan	1	1	1

TRABUQUETE

El trabuquete es un arma especial. Es un arma mecánica que lanza grandes proyectiles aprovechando la energía de un contrapeso colocado sobre el extremo corto de un brazo oscilante a cuyo otro extremo se coloca un péndulo con el proyectil.

En el lado del atacante existen 4 puntos en los que se pueden colocar los trabuquetes. En el lado del jugador solo se puede colocar 1 encima de la torre del homenaje siempre y cuando esta parte superior esté reparada (a través de mejoras). **El trabuquete permanece quieto, no puede avanzar.**

ARIETE

El ariete es una unidad que solo utiliza el atacante y que aparece por primera vez cuando las torres con el puente levadizo hayan caído y por lo tanto se desbloquee el camino hasta la puerta exterior del castillo. Esta unidad prioriza dirigirse tanto a esta puerta como a la puerta interior del castillo y solo ataca a estas construcciones, ignorando el resto de unidades.

MOZOS DE MURO

Estas unidades son exclusivas del defensor y se colocan entre los puestos de matacanes y buhedera, cuando se desbloquee la mejora correspondiente que habilita estos puestos. Tienen un alcance muy corto por lo que solo atacan a las unidades que estén justo debajo de ellos. Son especialmente efectivos contra los arietes.

MEJORAS

Las mejoras son una serie de ventajas que el jugador puede desbloquear entre rondas a costa de una gran cantidad de dinero. Existen tanto mejoras que aumentan las características de las armas de los soldados como mejoras que afectan al castillo o a la partida en general

MEJORAS DE ARMAS

Las mejoras de armas hacen que las unidades que portan el arma mejorada cambien su arma por el siguiente arma dentro de la misma categoría. Esta mejora afecta a todos los soldados que posean este arma mejorada. Por ejemplo, si se decide mejorar los arqueros y los arqueros tienen el Arco Compuesto, entonces todos pasarán a incorporar la Dactilera. Una vez adquirida una mejora de arma, todos los soldados que se compren con un arma de esa categoría tendrán esta mejora incorporada.

Los atacantes adquirirán estas mejoras con el paso de las rondas y según la dificultad de la partida.

BOTICA Y ENFERMERÍA

La Botica y la Enfermería son dos mejoras diferentes pero que surten el mismo efecto. Mejoran la curación de las unidades aliadas entre oleadas un 20% extra del porcentaje que se tiene antes de obtener cada mejora. Estas dos mejoras estarán disponibles desde el principio de la partida.

CALABOZO

La mejora del calabozo aumenta la cantidad de dinero obtenida tras superar las oleadas. En concreto aumenta un 20% el oro obtenido y se mantiene para las siguientes oleadas.

RASTRILLO

El Rastrillo aumenta la vida máxima y actual de las puertas de acceso a la muralla exterior e interior un 20%.

POZO DE SUMINISTRO

El Pozo de Suministro provoca que las unidades de artillería ligera se muevan más rápidamente por el subterráneo entre los diferentes puntos de ataque.

MEJORAS DE MURALLA

Las Mejoras de la Muralla aumentan un porcentaje de la vida máxima y de la vida actual de las murallas tanto exterior como interior del castillo. Existen 3 niveles de esta mejora y es necesario adquirir los primeros niveles para desbloquear los siguientes. El primer nivel aumenta un 10% estas cantidades, el segundo otro 10% y el tercero un 5%

RECONSTRUCCIÓN DE ARCOS DE LA TORRE DEL CABALLERO

La Reconstrucción de Arcos son también tres niveles de mejora. Para llegar al último nivel hay que haber comprado los dos anteriores. Cuando se completa el tercer nivel, se repara por completo la parte superior de la torre y se desbloquea la posibilidad de comprar el trabuquete, el cuál se coloca encima de ella.

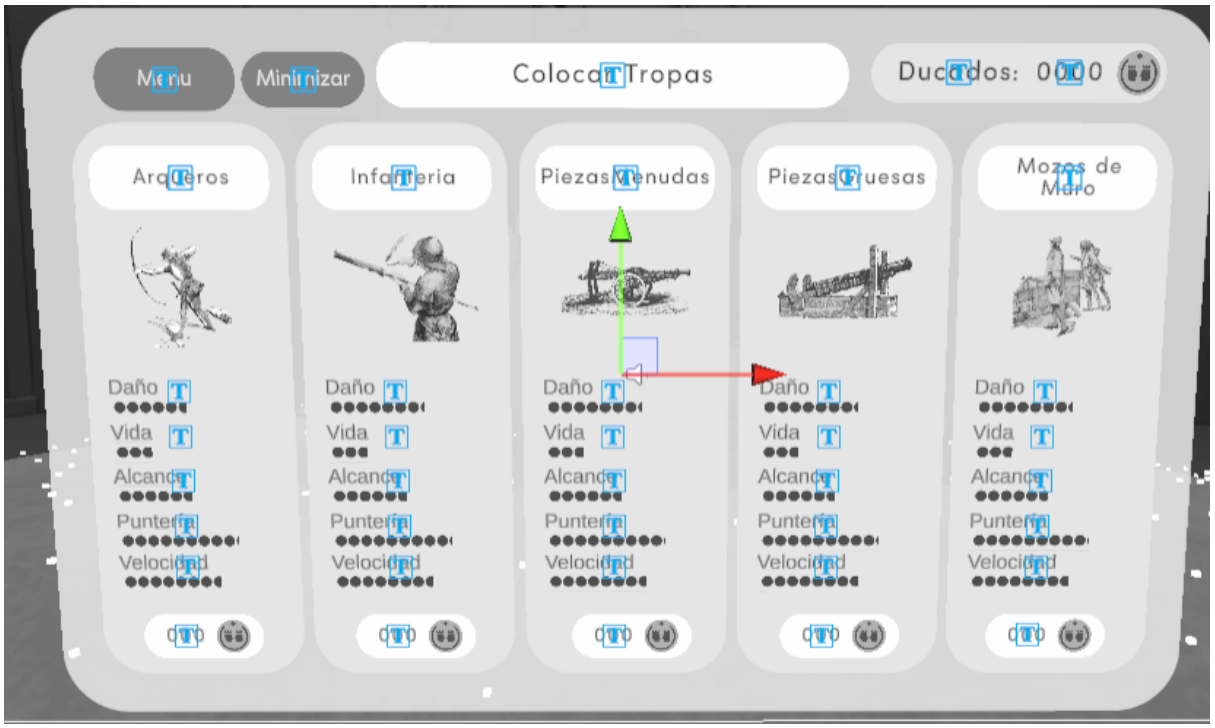
POSICIONES ESTRATÉGICAS

El jugador puede adquirir dos mejoras para desbloquear posiciones en el castillo que le ayuden con la defensa. Estas mejoras son los Matacanes y las Buhederas. Los Buhederas desbloquean dos posiciones que se encuentran encima de la puerta interior y los Matacanes varias posiciones en el castillo, entre ellas dos encima de la puerta exterior y varias en la torre del homenaje. También desbloquea la unidad Mozo de Muro, que es la única que puede situarse en estas posiciones.

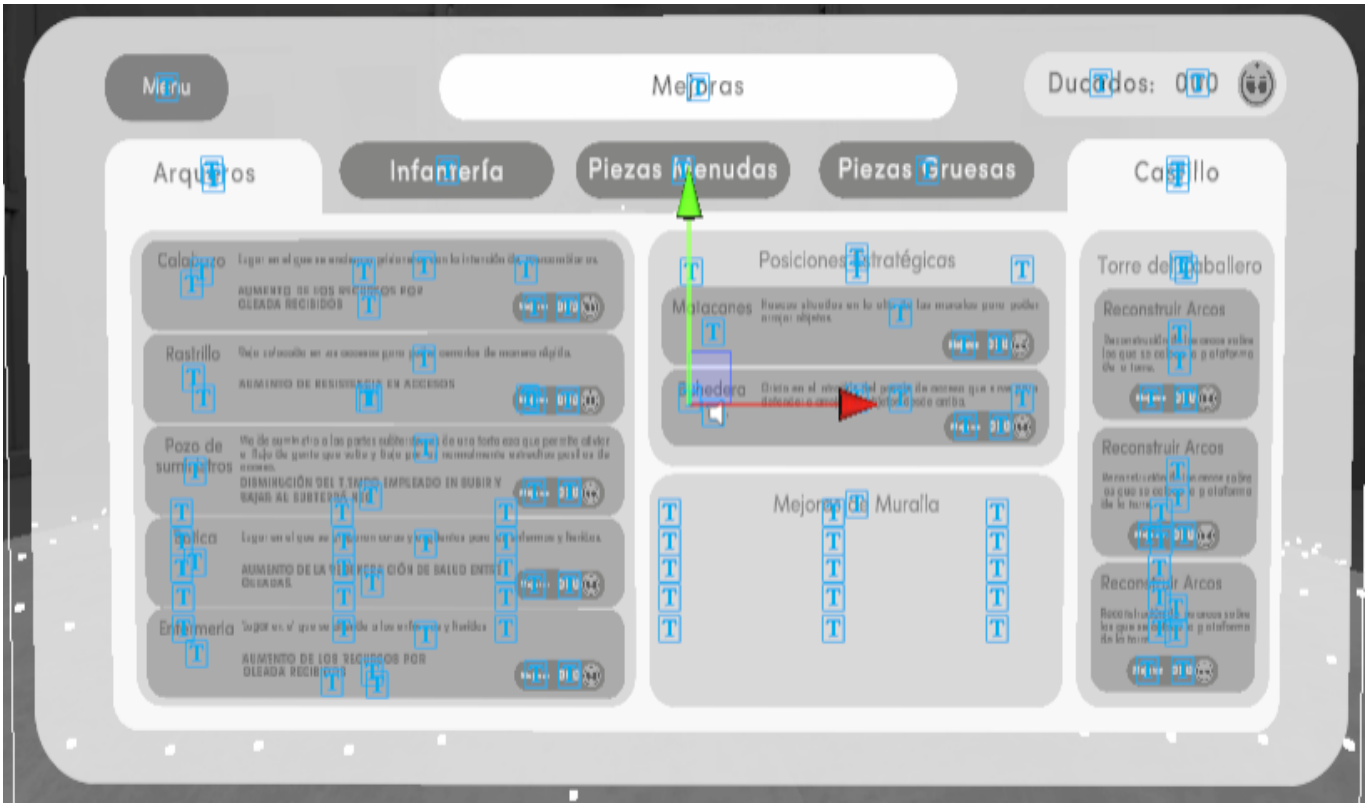
INTERFAZ DE USUARIO

MENÚ PRINCIPAL EN EL JUEGO

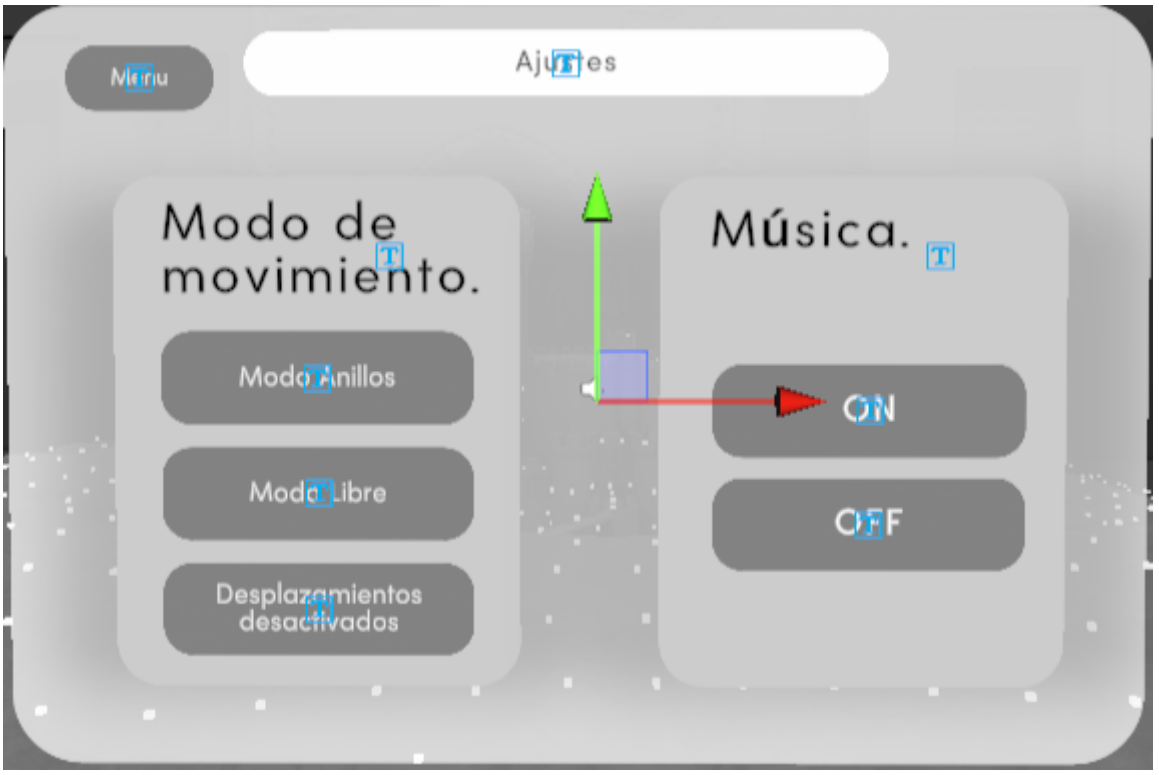
MENÚ DE TROPAS



MENÚ DE MEJORAS



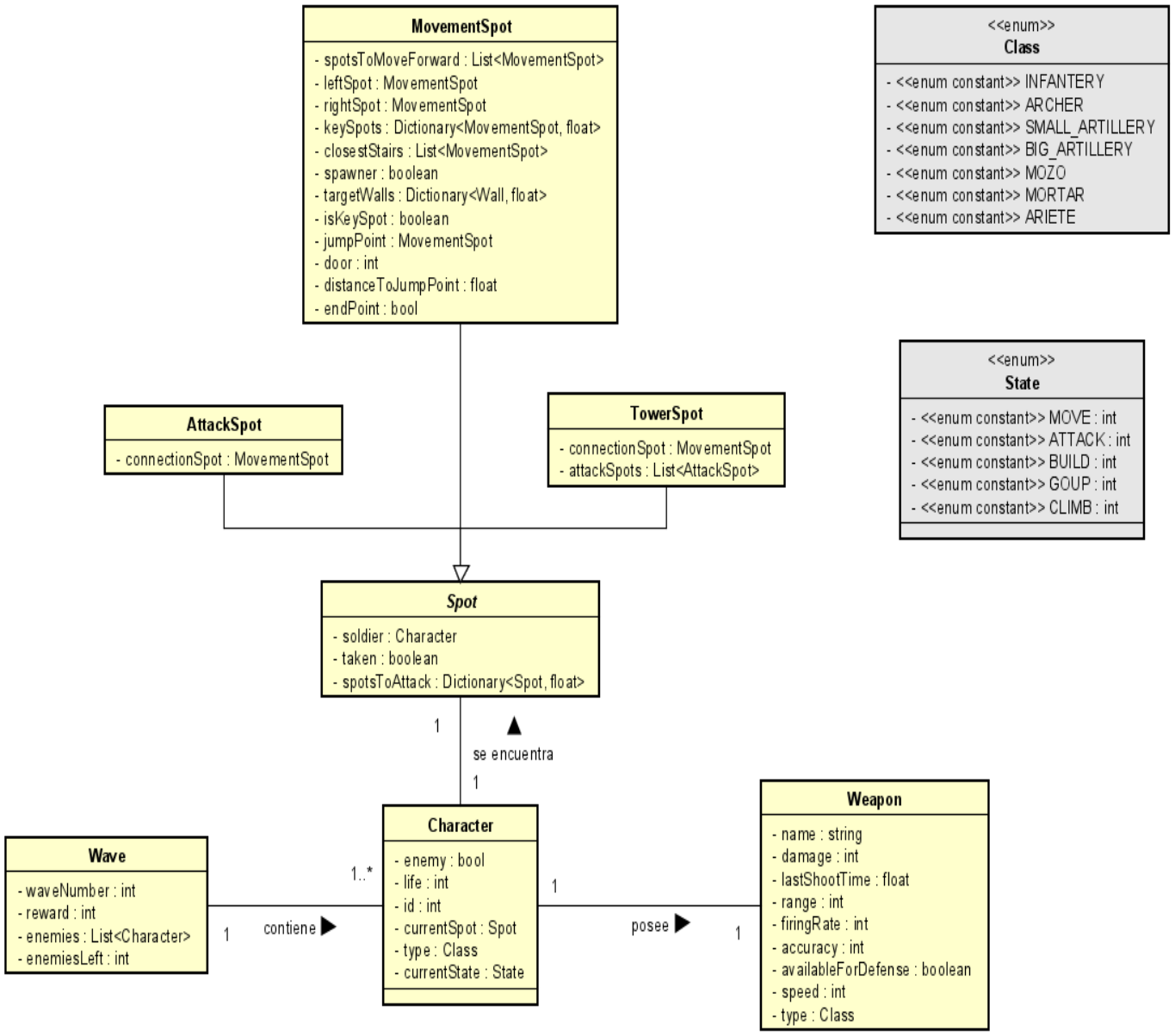
MENÚ DE AJUSTES



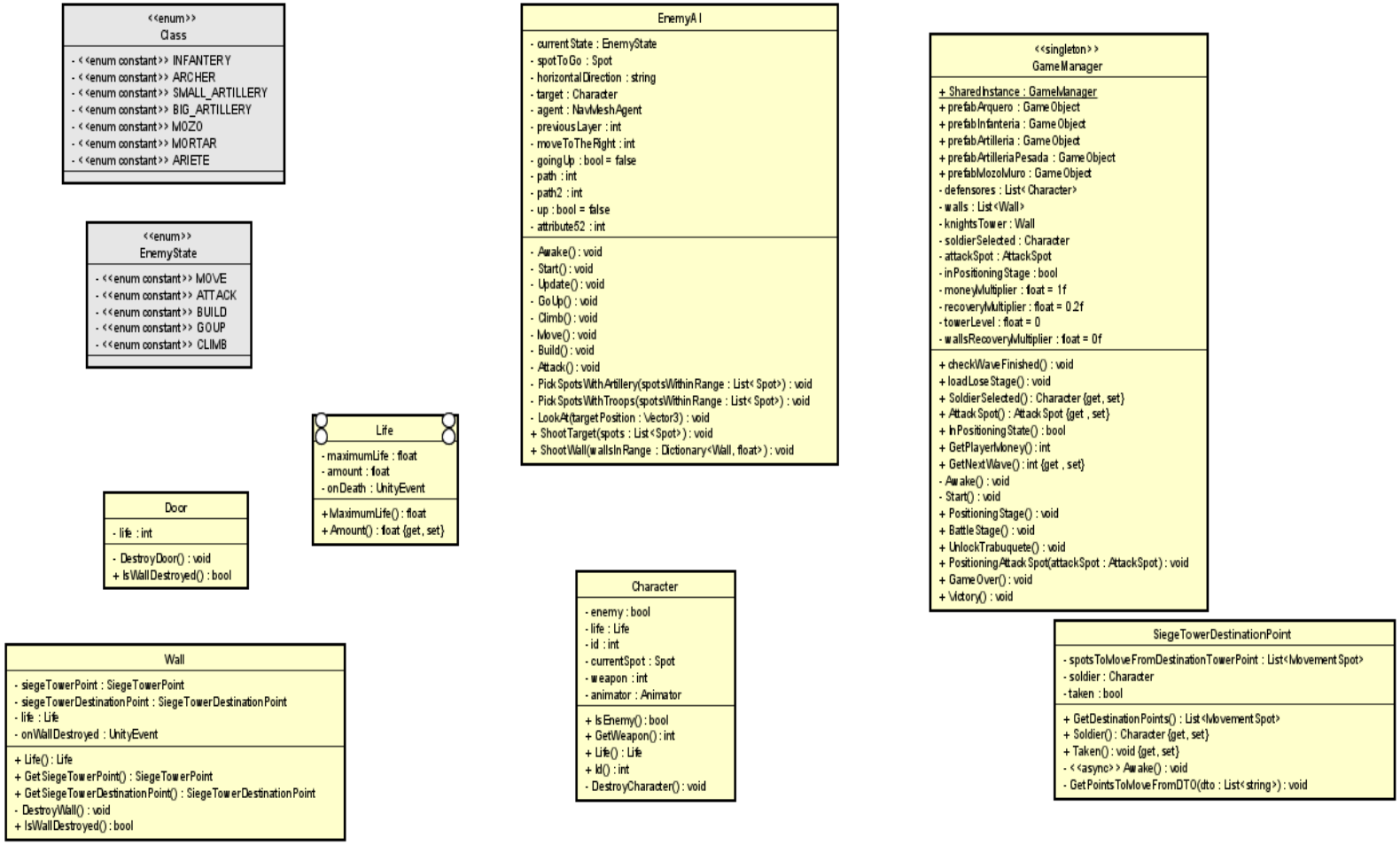
ELABORACIÓN DEL GDD

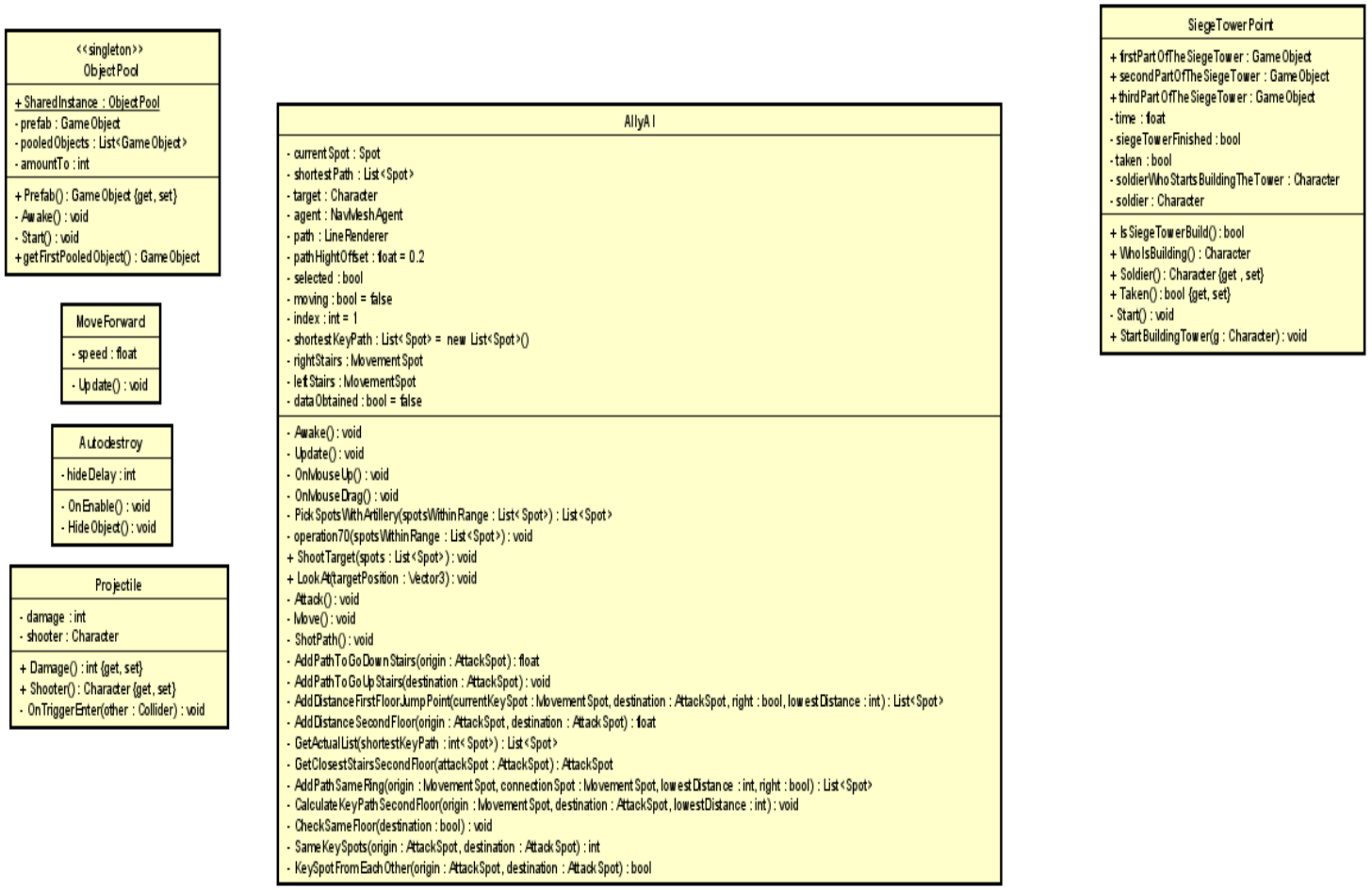
Para la elaboración del GDD se ha utilizado *Milanote*, un software libre para la organización de proyectos. *Aquí* está el proyecto. Se han utilizado [3] y [49] como referencias.

Apéndice II: Modelo de dominio inicial

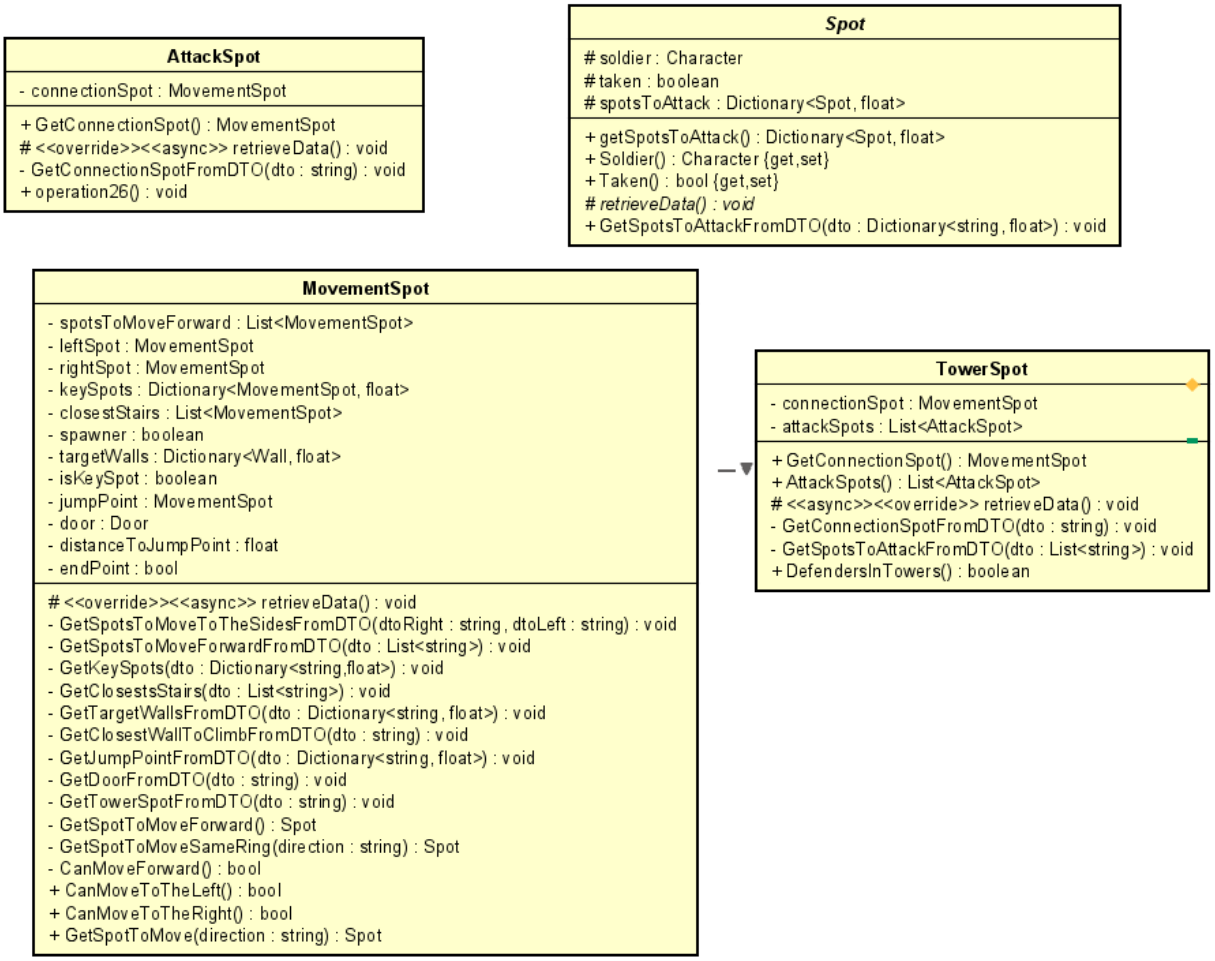


Apéndice III: Diagrama de diseño detallado

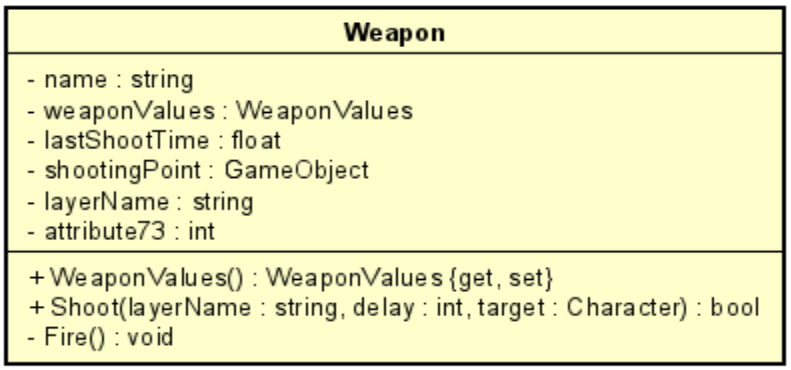


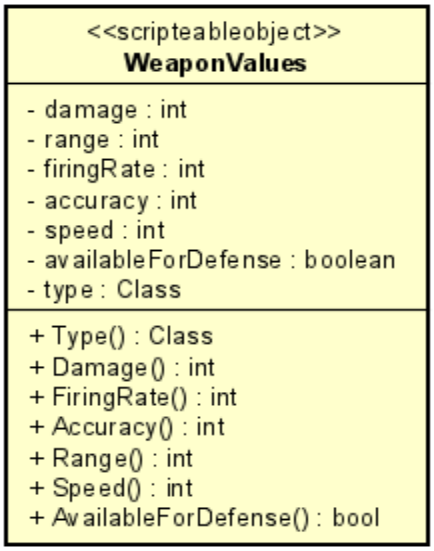

«singleton»
ObjectPool
+ SharedInstance : ObjectPool
- prefab : GameObject
- pooledObjects : List<GameObject>
- amountTo : int
+ Prefab() : GameObject {get, set}
- Awake() : void
- Start() : void
+ getFirstPooledObject() : GameObject
MoveForward
- speed : float
- Update() : void
Autodestroy
- hideDelay : int
- OnEnable() : void
- HideObject() : void
Projectile
- damage : int
- shooter : Character
+ Damage() : int {get, set}
+ Shooter() : Character {get, set}
- OnTriggerEnter(other : Collider) : void
AllyAI
- currentSpot : Spot
- shortestPath : List<Spot>
- target : Character
- agent : NavMeshAgent
- path : LineRenderer
- pathHightOffset : float = 0.2
- selected : bool
- moving : bool = false
- index : int = 1
- shortestKeyPath : List< Spot> = new List<Spot>()
- rightStairs : MovementSpot
- leftStairs : MovementSpot
- dataObtained : bool = false
- Awake() : void
- Update() : void
- OnMouseUp() : void
- OnMouseDrag() : void
- PickSpotsWithArtillery(spotsWithinRange : List< Spot>) : List<Spot>
- operation70(spotsWithinRange : List<Spot>) : void
+ ShootTarget(spots : List<Spot>) : void
+ LookAt(targetPosition : Vector3) : void
- Attack() : void
- Move() : void
- ShotPath() : void
- AddPathToGoDownStairs(origin : AttackSpot) : float
- AddPathToGoUpStairs(destination : AttackSpot) : void
- AddDistanceFirstFloorJumpPoint(currentKeySpot : MovementSpot, destination : AttackSpot, right : bool, lowestDistance : int) : List<Spot>
- AddDistanceSecondFloor(origin : AttackSpot, destination : AttackSpot) : float
- GetActualList(shortestKeyPath : int< Spot>) : List<Spot>
- GetClosestStairsSecondFloor(attackSpot : AttackSpot) : AttackSpot
- AddPathSameRing(origin : MovementSpot, connectionSpot : MovementSpot, lowestDistance : int, right : bool) : List<Spot>
- CalculateKeyPathSecondFloor(origin : MovementSpot, destination : AttackSpot, lowestDistance : int) : void
- CheckSameFloor(destination : bool) : void
- SameKeySpots(origin : AttackSpot, destination : AttackSpot) : int
- KeySpotFromEachOther(origin : AttackSpot, destination : AttackSpot) : bool
SiegeTowerPoint
+ firstPartOfTheSiegeTower : GameObject
+ secondPartOfTheSiegeTower : GameObject
+ thirdPartOfTheSiegeTower : GameObject
- time : float
- siegeTowerFinished : bool
- taken : bool
- soldierWhoStartsBuildingTheTower : Character
- soldier : Character
+ IsSiegeTowerBuild() : bool
+ WhoIsBuilding() : Character
+ Soldier() : Character {get , set}
+ Taken() : bool {get, set}
- Start() : void
+ StartBuildingTower(g : Character) : void

Apéndice IV: Diagrama de diseño detallado (paquete spots)

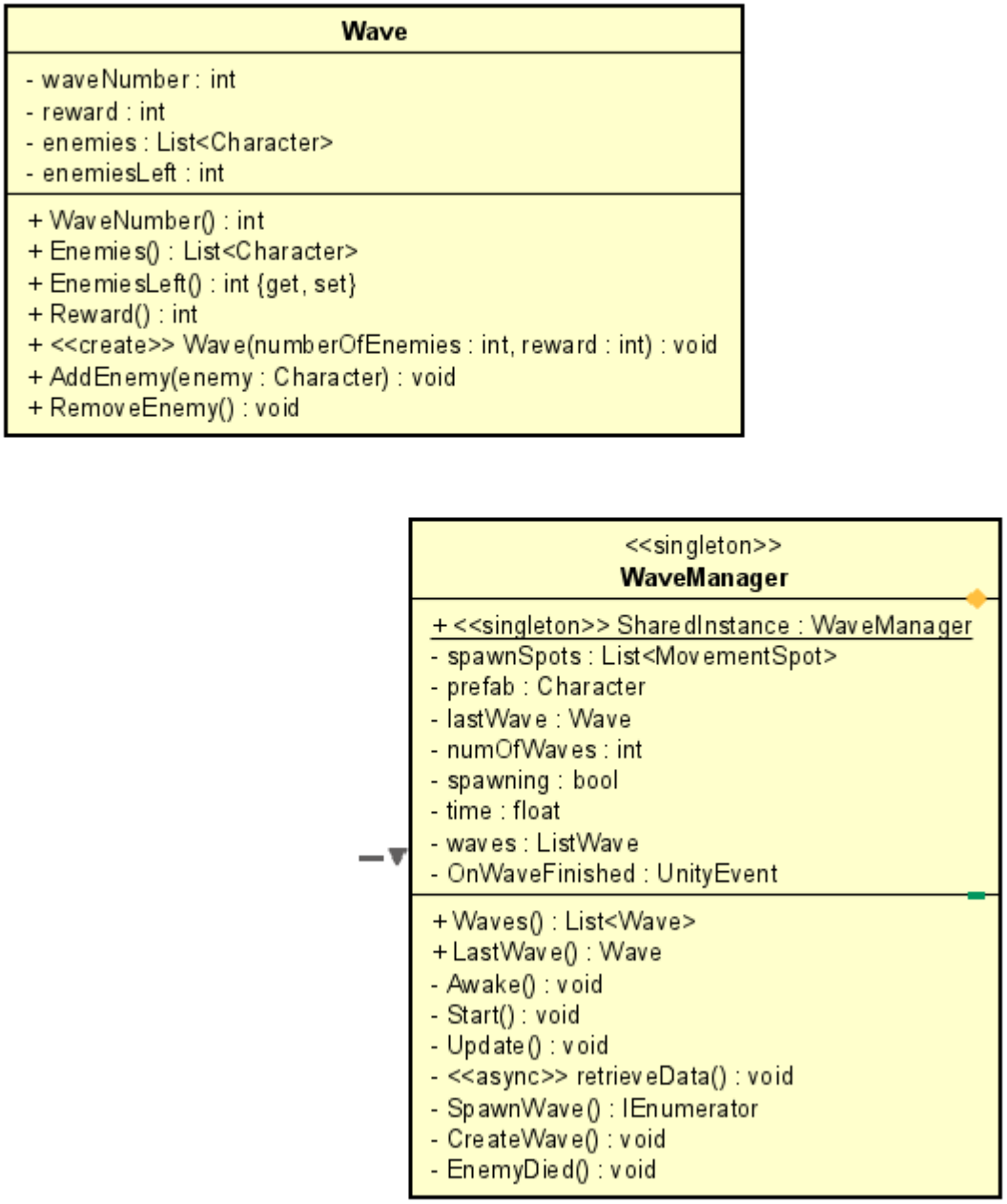


Apéndice V: Diagrama de diseño detallado (paquete weapons)





Apéndice VI: Diagrama de diseño detallado (paquete waves)



Apéndice VII: Anotaciones importantes

El proyecto solo funciona para ordenador, no para las gafas de realidad virtual.

Antes de arrancar, hay que quitar los scripts *DaoSpots* y *DaoWaves* del objeto *DAO* dentro del objeto *Managers* y volverlos a añadir al mismo objeto. Este es un bug no se ha sabido solucionar.

Bibliografía

[1] J.Clement. *Number of games released on Steam 2021*. (s/f). Statista. Recuperado el 12 de junio de 2022, de https://www.statista.com/statistics/552623/number-games-released-steam/

[2] Kirkcaldy, A. (2020, octubre 13). *Video game industry statistics, trends and data in 2022*. WePC | Let's Build Your Dream Gaming PC. https://www.wepc.com/news/video-game-statistics/

[3] D. Alfredo Fernández Ramos. "Desarrollo de videojuegos para la promoción del castillo de la Mota utilizando tecnología de realidad virtual", UVA, 2021.

[4] Aparicio, J. I. (2020, diciembre 14). *Los juegos más típicos de la Edad Media*. Historia del Condado de Castilla. https://www.condadodecastilla.es/blog/los-juegos-mas-tipicos-de-la-edad-media/

[5] *JUEGOS EN LA EDAD MODERNA*. (s/f). Blogspot.com. Recuperado el 12 de junio de 2022, de http://plusultra2015.blogspot.com/2015/04/juegos-en-la-edad-moderna.html

[6] Wikipedia contributors. (s/f). *Bertie the Brain*. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Bertie_the_Brain

[7] @ALVY (Twitter). 8 de Septiembre de 2021. "Videojuegos más antiguos que el más antiguo de los videojuegos". microservicios.com. https://www.microsiervos.com/archivo/juegos-y-diversion/videojuegos-mas-antiguos.html

[8] Wikipedia contributors. (s/f-b). *Historia de los videojuegos*. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/wiki/Historia_de_los_videojuegos#Década_de_1970

[9] CABRELLES SAGREDO, Ma Soledad. (s/f). *La influencia del juego para potenciar el desarrollo infantil en el ámbito educativo (I)*. Biblioteca Virtual Miguel de Cervantes. https://www.cervantesvirtual.com/obra-visor/la-influencia-del-juego-para-potenciar-el-desarrollo-infantil-en-el-ambito-educativo-i-783729/html/

[10] Hunicke, R., Leblanc, M., & Zubek, R. (s/f). *MDA: A formal approach to Game Design and game research*. Northwestern.edu. Recuperado el 13 de julio de 2022, de https://users.cs.northwestern.edu/~hunicke/MDA.pdf

[11] Hidalgo, I. P. (2018, abril 4). *TEORÍA DEL FLUJO o FLOW*. Psicólogos en Madrid. Hipnosis Clínica. Psicología Clínica. https://www.isidroperezhidalgo.com/teoria-del-flujo-o-flow/

[12] Vivaracho Pascual, Carlos Enrique. Apuntes Asignatura de Sistemas Multimedia, Tema 3 "El Jugador".. 2022

[13] Vivaracho Pascual, Carlos Enrique. Apuntes Asignatura de Sistemas Multimedia, Tema 1 "El Juego".. 2022

[14] Antonio, M. (s/f). *¿Qué pruebas debemos hacerle a nuestro videojuego?· Antonio Moon´s*. Github.io. Recuperado el 13 de julio de 2022, de https://moonantonio.github.io/post/2018/comun/004/

[15] Realidad Virtual. (s/f). Rae.es. Recuperado el 13 de julio de 2022, de https://enclavedeciencia.rae.es/realidad%20virtual

[16] *REALIDAD VIRTUAL*. (s/f). Deusens.com. Recuperado el 13 de julio de 2022, de https://deusens.com/es/blog/hitos-historia-realidad-virtual

[17] Ferrario, S. (s/f). *Historia de la Realidad Virtual – Xperimenta Cultura*. Xperimentacultura.com. Recuperado el 13 de julio de 2022, de https://xperimentacultura.com/historia-de-la-realidad-virtual/ç

[18] Facebook.com. Recuperado el 13 de julio de 2022, de https://store.facebook.com/es/quest/products/quest-2/

[19] García, J. (2020, octubre 17). *Oculus Quest 2, análisis: una de las mejores (y asequibles) opciones para iniciarse en la realidad virtual*. Xataka.com; Xataka. https://www.xataka.com/analisis/oculus-quest-2-analisis-caracteristicas-precio-especificaciones

[19] López, P. (2020, septiembre 15). *Aparecen las Oculus Quest 2 con un Snapdragon XR2 y pantallas casi 2K*. GEEKNETIC. https://www.geeknetic.es/Noticia/19998/Aparecen-las-Oculus-Quest-2-con-un-Snapdragon-XR2-y-pantallas-casi-2K.html

[21] Martínez, C. (2020, abril 17). *Los mejores juegos de realidad virtual para tus Meta Oculus Quest*. ElOutput. https://eloutput.com/videojuegos/listas/mejores-juegos-realidad-virtual-oculus-quest/

[22] *The walking dead – saints & sinners*. (s/f). Vrwalkingdead.com. Recuperado el 25 de abril de 2022, de https://www.vrwalkingdead.com

[23] Márquez, R. (2020, septiembre 27). *Vader Immortal es la excusa perfecta para pedir una evolución en la PlayStation VR de PS5*. Vidaextra.com; Vida Extra. https://www.vidaextra.com/analisis/vader-immortal-excusa-perfecta-para-pedir-evolucion-playstation-vr-ps5

[24] *Crisis VRigade 2: ANÁLISIS*. (s/f). Realovirtual.com. Recuperado el 28 de abril de 2022, de https://www.realovirtual.com/articulos/5930/crisis-vrigade-2-analisis

]25] *Larcenauts: ANÁLISIS*. (s/f). Realovirtual.com. Recuperado el 13 de julio de 2022, de https://www.realovirtual.com/articulos/5825/larcenauts-analisis

[26] *Beat Saber*. (2022, enero 20). 🥽 VR Juegos. https://vrjuegos.net/beat-saber-cortar-bloques-al-ritmo-de-la-musica/

[27] *Half-Life: Alyx*. (s/f). Half-Life. Recuperado el 13 de julio de 2022, de https://half-life.com/es/alyx

[28] Univision, P. (2015, mayo 18). *Defiende tu torre con los 7 mejores tower defense de la historia*. Univision. https://www.univision.com/entretenimiento/cultura-pop/defiende-tu-torre-con-los-7-mejores-tower-defense-de-la-historia

[29] Fariñas, A. (2021, marzo 18). *Mejores 9 juegos Tower Defense para Android (2022): online, gratis, cooperativos*. Gamovil. https://gamovil.com/juegos/mejores-juegos-android/mejores-juegos-tower-defense-android

[30] Torrejón, R. (2021, agosto 14). *Orcs Must Die! 3 me ha recordado lo divertido que es matar orcos: un tower defense que va de menos a más*. 3djuegospc.com; 3DJuegos PC. https://www.3djuegospc.com/analisis/orcs-must-die-3-me-ha-recordado-divertido-que-matar-orcos-tower-defense-que-va-a

[31] *Dungeon Defenders: Awakened, review: la histórica torre de defensa está de vuelta en el campo*. (2021, octubre 29). ➤Las Mejores Noticias y Guías para tus Videojuegos, Comics y Películas | Resources4Gaming ®.

https://www.resources4gaming.com/es/dungeon-defenders-awakened-review-la-historica-torre-de-defensa-esta-de-vuelta-en-el-campo

[32] Darkor_LF. (2017, octubre 17). *Kingdom Rush Frontiers, un cautivador tower defense*. Todas Gamers. https://todasgamers.com/2017/10/17/kingdom-rush-frontiers-cautivador-tower-defense/

[33] Hero, T. W. (2021, diciembre 16). *Realm Defense: Leyenda heroica*. ⚔️ Defiende la Torre 🏹. https://defiendelatorre.com/juegos/realm-defense-hero-legends-td/

[34] *Radian Defense, tower defense modernizado*. (2014, agosto 3). Eljugondemovil.com. https://www.eljugondemovil.com/radiant-defense-dandole-una-vuelta-los-tower-defense/

]35] Channel, D. [disneychannel]. (2019, marzo 15). *Baymax dreams | compilation | big hero 6 the series | Disney channel*. Youtube. https://www.youtube.com/watch?v=Q2XEyCFAMuk

[36] España, D. [disneyspain]. (2019, abril 10). *El Rey León (2019) | Tráiler Oficial en español | HD*. Youtube. https://www.youtube.com/watch?v=mb79ctR-E-c

[37] Unity Technologies. (s/f). *Unity*. Unity. Recuperado el 24 de mayo de 2022, de https://unity.com

[38] Candil, D. (2014, febrero 21). *Unity, el motor de desarrollo capaz de partir la historia de los videojuegos en dos*. Vidaextra.com; Vida Extra. https://www.vidaextra.com/industria/unity-el-motor-de-desarrollo-capaz-de-partir-la-historia-de-los-videojuegos-en-dos

[39] Unity Technologies. (s/f-b). *Unity user manual 2021.3 (LTS)*. Unity3d.com. Recuperado el 27 de mayo de 2022, de https://docs.unity3d.com/Manual/index.html

[41] Instituto Tecnológico de Nuevo Laredo. Recuperado el 2 de julio de 2022, de http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/A-Star/A-Star(2005-II-A).pdf

[41] *Pathfinding: A*. (2014, abril 12). lambda lab 85. https://www.lanshor.com/pathfinding-a-estrella/

[42] Unity Technologies. (s/f). *Comparar planes de Unity: Plus, Pro y versión gratuita. ¡Elige el mejor motor 2D o 3D para tu proyecto!* Unity Store. Recuperado el 3 de julio de 2022, de https://store.unity.com/es/compare-plans

[43] *.NET: Qué es y cómo funciona*. (2020, junio 15). aula21 | Formación para la Industria; aula21. https://www.cursosaula21.com/que-es-net/

[44] Smacchia, P. (2022, junio 28). *Visual Studio vs. Jetbrains Rider performance*. NDepend. https://blog.ndepend.com/visual-studio-vs-jetbrains-rider-performance/

[45] *Principios del Manifiesto Ágil*. (s/f). Agilemanifesto.org. Recuperado el 14 de julio de 2022, de https://agilemanifesto.org/iso/es/principles.html

[46] *¿Cuánto Cobra un Ingeniero? (Sueldo 2022)*. (s/f). Jobted.es. Recuperado el 10 de julio de 2022, de https://www.jobted.es/salario/ingeniero

[47] Unity Technologies. (s/f). *Introduction to Object Pooling*. Unity Learn. Recuperado el 6 de julio de 2022, de https://learn.unity.com/tutorial/introduction-to-object-pooling

[48] Parra, D. (2022, marzo 17). ➢ *Patrones de diseño - Object Pool - 🎮 The power ups 💪*. 🎮 The power ups 💪. https://thepowerups-learning.com/patrones-de-diseno-object-pool/

[49] *silent-hill-design-document.pdf*. (s/f). Google Docs. Recuperado el 7 de julio de 2022, de https://drive.google.com/file/d/1nxvdXasP-HsRCt62cHK3wF_pIrJpYx5T/view