Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un tower defense 2D en Unity

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Bernácer Ferrer, Martín

Tutor: Lluch Crespo, Javier

2018-2019

Resumen

Este proyecto desarrolla un videojuego 2D del género *tower defense* con el motor *Unity3D*. El videojuego se pensó para el concurso de minijuegos.com, por lo que el objetivo del diseño fue ser jugado en un navegador y su compilación se realizó en WebGL.

El videojuego consiste la defensa de una puerta con la ayuda de tres armas, que cada una de ellas dispone de oportunidad de mejora. Tres tipos de enemigos diferentes intentarán destruirla y, dependiendo de su distribución, el jugador deberá emplear la mejor arma para las situaciones que se produzcan en cada ataque.

Palabras clave: tower defense, Unity, Unity 3D, videojuego, minijuego, minijuegos.com, shooter, Door Defender, 2D.

Abstract

This project develops a 2D tower defense videogame with the Unity 3D engine. The game is designed for the minijuegos.com contest, so it will be played in browser and it is compiled in WebGL.

The videogame consists of defending a door using three different weapons, which can be upgrade. Three types of enemies will try to destroy it, and depending on the distribution of the enemies, the player has to choose the best weapon for that situation.

Keywords : tower defense, Unity, Unity 3D, videogame, minigame, minijuegos.com, shooter, Door Defender, 2D.

Resum

Aquest projecte desenvolupa un videojoc 2D del gènere *tower defense* amb el motor Unity 3D. El videojoc està pensat per al concurs de minijuegos.com, pel que està dissenyat per a ser jugat en un navegador i la seva compilación es realitza en WebGL.

El videojoc consisteix a defensar una porta amb l'ajuda de tres armes, les quals és possible millorar. Tres tipus d'enemics diferents intentaran destruir-la, i depenent de la distribució d'enemics, el jugador haurà d'emprar la millor arma per a aquesta situació.

Paraules clau : tower defense, Unity, Unity 3D, videojoc, minijoc, minijuegos.com, shooter, Door Defender, 2D.

Agradecimientos

A Paco Fuster Ferrer, por su gran ayuda en el desarrollo del contenido gráfico de este juego.

A mi tutor, Javier Lluch Crespo, por su trabajo en la supervisión de este proyecto.

Índice de contenidos

Índice de imágenes

1. Introducción

Los videojuegos constituyen una de las industrias que más dinero mueven en el mundo, y su público aumenta cada año. Según el último informe de la Asociación Española de Videojuegos, esta industria facturó en España 1.530 millones de euros en 2018, un 12,6% más que en 2017, siendo la primera del ramo audiovisual, por encima del cine (585,7 millones) y la música grabada (237,2 millones) [1].

Desde el puro entretenimiento hasta la posibilidad de encarnar personajes totalmente ajenos al jugador y experimentar sus vivencias, los videojuegos poseen una característica que ningún otro medio consigue: la interactividad.

Aunque la idea de poder transmitir emociones al jugador es loable, desarrollar un juego con esas capacidades requeriría demasiado tiempo, por lo que en este proyecto se escogió la opción del entretenimiento, en concreto un juego del género *tower defense*, pero hibridado con un *shooter*.

De todos modos, no debemos menospreciar el valor del entretenimiento. Jugar es una actividad primitiva, muchas especies animales lo realizan: su origen y sentido se encuentra en su propio código genético. El juego es un mecanismo que pone a prueba las habilidades y las perfecciona. Los videojuegos entrenan muchas capacidades cognitivas y de estrategia social sin que la persona sea consciente de ello, y proporcionan ayuda para que sea más sencillo para el individuo la realización de actividades fuera de ellos, a lo que se debe añadir la capacidad educativa y rehabilitadora que muchos videojuegos poseen [2].

El juego desarrollado, denominado **Door Defender**, consiste en conseguir la derrota de unas máquinas con aspecto de artrópodos para que no destruyan la puerta que se tiene como misión proteger. Para cumplir ese objetivo, el jugador dispone de tres armas:

- Un cañón ligero y rápido, pero con escasa capacidad de daño.
- Un arma láser, de actuación lenta, pero con la capacidad de traspasar a los enemigos con un grado medio de provocación de daño.
- Un lanzacohetes, de gran capacidad destructiva y con área de efecto, aunque con la característica de ser el armamento que exhibe una mayor lentitud.

A medida que el jugador va derrotando enemigos, estos abandonarán chatarra que permitirá al jugador mejorar sus armas y la resistencia de la puerta. Además, la puerta viene equipada con un escudo, que proyecta una barrera defensiva que absorbe cualquier proyectil enemigo durante unos segundos y que para conseguir reutilizarlo se debe esperar un tiempo tras su activación.

Normalmente, los *tower defense* utilizan torretas automáticas, que pueden ser objeto de mejora. En este caso, el jugador es la única torre, y no existe ningún elemento automático para la defensa.

El juego ha sido desarrollado en Unity, utilizando C# para los *scripts*, y será compilado en WebGL, ya que era un requisito para participar en el concurso de *minijuegos.com*.

En la asignatura de Sistemas Gráficos Interactivos se propuso realizar un trabajo de fin de grado sobre el desarrollo de un videojuego en Unity que además participaba en un concurso donde los mejores serían premiados y publicados en la página de *minijuegos.com*, la web de juegos en castellano que presenta un tráfico mayor.

En este documento se presentará la evolución de los videojuegos como medio y sus ramificaciones a lo largo de los últimos años, diversificándose en multitud de géneros que han seguido un proceso de depuración e hibridación, generando modalidades de juego nuevas mediante *mods* (modificaciones no oficiales), modificando totalmente la jugabilidad del juego original.

Más adelante, se analizará los *tower defense* más exitosos y las características por las que el juego creado en este proyecto se diferencia de ellos, inspirándose en sus virtudes y eliminando aquellos elementos que restan a la experiencia.

A continuación, se describirá detalladamente los aspectos más relevantes del motor utilizado, Unity3D, y algunos de los elementos de su editor más importantes.

Posteriormente, habrá una referencia al análisis y diseño realizado antes de comenzar el proyecto y todos los detalles que se fueron modificando mientras se desarrollaba, elementos que perdían sentido tras reflexionar sobre ellos o al observarlos en acción.

Seguidamente, se desarrollará la implementación, con el relato del proceso de creación del proyecto, desde las pequeñas piezas que pueden funcionar por sí solas hasta la gran estructura de clases relacionadas entre sí que dan vida al videojuego final.

Tras la implementación llegarán las pruebas: el juego fue ensayado por diferentes usuarios, y las respuestas recibidas modificaron el producto para mejorar la experiencia de los jugadores que lo consumieran más adelante.

Al final, se mostrará el resultado final del juego, las conclusiones obtenidas de la realización del proyecto y todos los detalles que podrían mejorar el videojuego desarrollado, que por tiempo u otros motivos no pudieron llevarse a cabo.

Se ha añadido un glosario al final del trabajo, dado que la terminología empleada puede resultar muy específica de la industria del videojuego y no tanto de términos informáticos.

1.1 Motivación

La principal motivación en la realización de este proyecto es ganar experiencia para poder producir videojuegos cada vez más complejos. Con esta idea ya en mente, se ofreció la oportunidad de hacer un TFG que además participaría en un concurso donde, si el juego fuera lo suficiente bueno, se publicaría en su página web. El concurso exigía que se desarrollará en Unity, lo cual no fue un problema porque coincidía con la idea original de su elaboración.

2. Objetivos

El objetivo ha sido desarrollar un juego simple pero entretenido, que case con el mercado de minijuegos.com, pero que incluyera de forma superficial todos los elementos que caracterizan a un juego de mayor dimensión.

Para conseguir el objetivo principal, la primera misión debía ser conseguir una fluidez significativa en la utilización de las armas. Si nuestra mecánica de juego principal, y prácticamente único elemento para poder interactuar con el entorno, no resultara satisfactoria, el jugador no querría continuar jugando y abandonaría el juego. Para evitarlo, la jugabilidad deberá de ser lo suficientemente agradable como para que se pueda permanecer en acción una hora o dos seguidas sin generar molestia.

Una vez establecido que el procedimiento de disparo sea lo suficientemente entretenido, hay que plantear un objetivo con el que utilizar esta habilidad: los enemigos. El enemigo tiene que ser una amenaza: si consigue llegar hasta la puerta, la vida de esta descenderá rápidamente. Pero no solo es importante que sea una amenaza. El jugador, además, debe tener claro cuándo sus ataques han sido relevantes contra el enemigo.

Los efectos visuales pueden parecer irrelevantes, pero no es así. Unos buenos efectos especiales pueden tener un gran impacto en la motivación del jugador.

Dar al jugador una opción defensiva es una gran forma de introducir un elemento estratégico y añadir una capa de complejidad al juego. Un enemigo duro que realice un ataque muy poderoso y que el jugador no disponga de ninguna opción, al enfrentarse a un enemigo que posee un índice de vida demasiado alto, puede resultar frustrante. Conseguir evitar ese daño, por una vez, relaja un poco la dificultad, pero con un tiempo prolongado de reutilización puede regresar al jugador a un estado de frustración.

Pero el verdadero objetivo del juego es lograr que el jugador se sienta poderoso. Todo lo expresado con anterioridad son capas de “mentiras” para llegar a ese fin. Si se ofrece la posibilidad de mejorar las armas, se alcanzará el punto en el que ese enemigo, que tantas balas requirió para conseguir eliminarlo, podrá ser derrotado con un solo proyectil.

Cuanto más amenazador es el enemigo, mayor es la satisfacción al derrotarlo. Añadir nuevos enemigos más duros en fases superiores contribuirá a la sensación de poder del jugador.

Y, para terminar, como clímax a la incesante cantidad de enemigos, llegará el más poderoso, un jefe final. Muy superior a cualquier otro enemigo pero que, al ser derrotado por el jugador, la horda de enemigos terminará. Y el jugador habrá acabado el juego.

En definitiva, los objetivos del desarrollo de este videojuego para poder lograr los elementos descritos arriba, que dan sentido a la interactuación con el jugador, se pueden resumir en los siguientes puntos:

- Implementación de tres armas, cada una para una situación diferente.
- Desarrollo de un arma ligera.
- Desarrollo de un arma láser.
- Desarrollo de un arma pesada.
- Implementación de tres enemigos con patrones de ataque distintos.
- Ofrecimiento de una opción defensiva.
- Mecanismos de mejora de las armas.
- Implementación de tres niveles con incremento progresivo en dificultad.
- Implementación de un jefe final.

3. Estado del Arte

Como ya se indicó al comienzo de la introducción, los videojuegos constituyen una de las industrias más potentes a nivel mundial, no sólo a nivel de entretenimiento, sino también en relación con otros sectores [3] Moviliza enormes recursos económicos, cerca de 100.000 millones de dólares de facturación en 2018, y también recursos humanos, con centenares de miles de profesionales implicados a nivel mundial. También comprometen áreas de conocimiento de tecnología, diseño, creación de guiones, cinemática, música…y en los últimos años se han caracterizado por una gran repercusión e influencia en las redes, con incremento de los juegos multijugador y el desarrollo de competiciones a nivel mundial que han conseguido completar el aforo de estadios deportivos en la celebración de las finales, con importante repercusión en los medios [4].

Por otro lado, se han establecido múltiples foros internacionales para la convergencia de todos los actores de esta industria, destacando la feria E3 (Electronic Entertainment Electronic [5].

Desde 2008, se ha señalado el día 29 de agosto como el Día Mundial de los Videojuegos.

[Figura 1] y [Figura 2] pueden ilustrar la realidad actual del impacto económico de esta industria [6].

Figura 1: Impacto económico de la industria de videojuegos

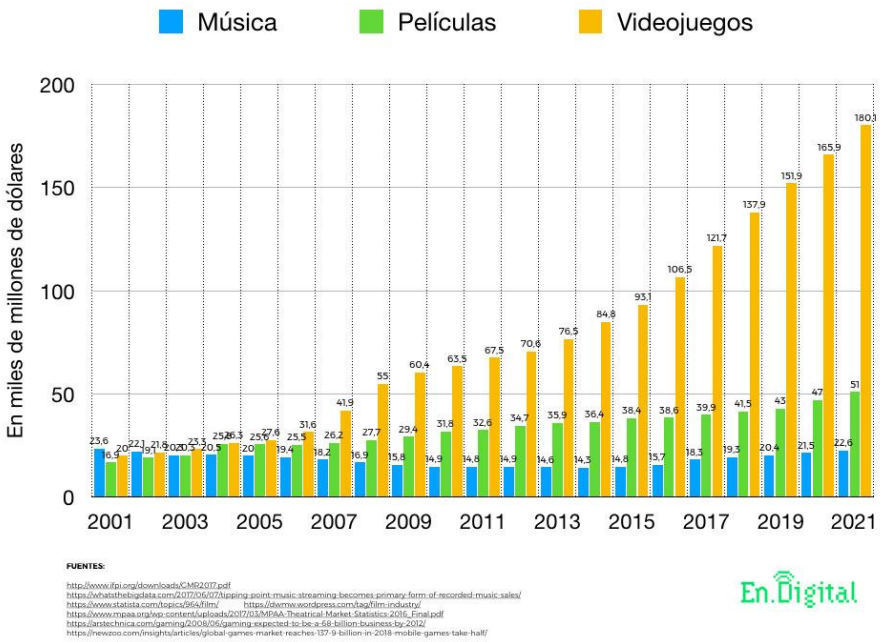


Figura 2: Grafico de comparación de la industria de Música, Películas y Videojuegos

Sin embargo, el mundo del videojuego es poliédrico y trasversal, trascendiendo al mero entrenamiento para llegar a interactuar con elementos básicos de nuestra psicología y entidad cultural: emociones, creatividad, narrativa, poesía, crítica y estrategia social, surrealismo, humor, significado de la guerra, anticipación, filosofía, enfrentamiento con los arquetipos…

Y como ya se ha mencionado en la introducción, hay que considerar el elemento más novedoso y atractivo que poseen los videojuegos: la interactividad.

Resultado de esta importancia, en el año 2009 se aprobó por unanimidad una Proposición No de Ley (PNL)» del Congreso de los Diputados, con el siguiente texto:

«La Comisión de Cultura del Congreso de los Diputados establece que el videojuego constituye un ámbito fundamental de la creación y la industria cultural de España. En consecuencia, insta al Gobierno a reconocer a sus creadores y emprendedores como protagonistas de nuestra cultura. Asimismo, en el marco de sus competencias y en coordinación con las administraciones autonómicas competentes, se insta al Gobierno a facilitar su acceso a todas las ayudas factibles para la promoción de su actividad, la financiación como industria cultural y la internacionalización de sus iniciativas» [7].

En este sentido, para recalcar la importancia actual del videojuego en el mundo global, conviene señalar también que el primer ministro japonés en 2016 presentó disfrazado de un personaje de videojuego, Super Mario de Nintendo, el anuncio en Río de Janeiro que la nueva edición de los juegos olímpicos se celebraría en Tokio en el año 2020 [Figura 3] [8].

Figura 3: Anuncio de JJOO Tokio 2022 en Rio de Janeiro

La historia de los videojuegos comienza en la década de los años sesenta. Aunque es un tema muy debatido, se suele otorgar el título de "primer videojuego de la historia" a *Tennis for Two* , creado en 1958 por Willy Higinbotham , que empleó un osciloscopio para su realización. Pero, realmente, el primero que destacó fue el videojuego *Spacewar!*, de 1961, un juego con física realista creado por Wayne Wiitanen, Steve Russell y Martin Graetz que consistía en una batalla de dos naves capaces de dispararse entre sí. Fue el primer *shooter* de la historia.

No hay que olvidar que para que surgiera el concepto videojuego, fue necesario que determinados hechos dentro del campo de la ciencia y de la cultura del entretenimiento se hubieran desarrollado en los años previos. En concreto, se trata de los avances en computación, en imagen de vídeo, pero también en el auge de la cultura *pulp* y el interés por la ciencia ficción, así como la existencia de determinados empresarios que decidieron lanzarse a este novedoso campo con perspectiva de rentabilidad económica. En definitiva, los videojuegos deben mucho a Alain Turing, Ralph Bauer, al grupo tecnológico de Massachusetts y a Nolan Bushnell, cofundador de la compañía Atari.

Esta compañía creó un juego de máquina recreativa de moneda llamado Pong en 1972, uno de los videojuegos más famosos de la historia, que se basaba en el tenis de mesa. Más tarde, Atari produjo una videoconsola solo para el videojuego Pong, la cual impulsó a esta empresa, convirtiéndose en una de las más exitosas de esa época.

Estos juegos se desarrollaron en máquinas recreativas y en consolas individuales primitivas. Aparecieron múltiples locales para jugar a videojuegos que también cumplieron una función de centros de relación social.

En aquellos tiempos, el jugador buscaba la sensación kinestésica, la comprensión del espacio simulado y el desarrollo de su habilidad como jugador. Estas características,

que no han desaparecido hoy en día cuando un jugador se enfrenta a un videojuego, comenzaron a experimentarse en las máquinas *arcade* y en las primeras videoconsolas con extensión comercial, como la Odyssey de 1972 y la Atari 2600 de 1977 [Figura 4].

Figura 4: Atari 2600

En los años setenta, hubo una verdadera explosión del fenómeno del videojuego. Además de los citados, aparecieron otros juegos que hoy en día se consideran clásicos como el Space Invaders de Tomohiro Nishikado (el típico juego de “matamarcianitos”), que supuso una novedad en la jugabilidad pues permitía progresión y aprendizaje y disfrute añadido del usuario. También hay que destacar el videojuego Pacman [Figura 5], que en España se denominó “comecocos”.

Figura 5: Space Invader y PAC-MAN

En Estados Unidos, estos juegos se podían encontrar hasta en la sala de espera de las consultas médicas, circunstancia que todavía se sigue empleando en Japón [9].

En esta década también nacieron los géneros, que posteriormente se fueron extendiendo en distintas ramas según sus características y deseos de los jugadores. En este punto se debe destacar que los videojuegos en sí no disponen de un registro de propiedad intelectual. En la década de los ochenta, fue famoso un procedimiento judicial en los EE. UU. que produjo jurisprudencia en este sentido. Por esta razón, la pertenencia a un subgénero y el plagio caminan por una senda muy estrecha.

Ejemplos de estos géneros que fueron creciendo: arcade, puzles, plataformas, simuladores de vuelo, aventura, romance, RPG y sus variantes, de rol… Esta variedad permitió el disfrute de diferentes entornos y la utilización de habilidades diferentes.

Ya al final de los setenta, surgió la primera crisis que exigió al videojuego un cambio de paradigma. Atari, la mayor empresa hasta el momento, quebró por múltiples causas, entre ellas la mala gestión económica, la falta de visión y la saturación del mercado, y en esas circunstancias el relevo lo tomó Europa, en concreto Gran Bretaña, con la aparición de determinados microrordenadores personales y las posibilidades de jugar con un teclado y de compra de distintos juegos, que ya se ha había ofrecido a las primitivas consolas individuales (aunque la mayoría solo admitía jugar a los elementos incluidos en su software) pero que en este momento se hizo masivo y permitió el crecimiento de esta industria. Aunque existían otros modelos, Sinclair creó el Spectrum ZX80 [Figura 6], que permitirá, además de jugar, realizar muchas otras actividades: entre ellas, programar. Otros sistemas simultáneos en aquel tiempo fueron Amstrad, el Commodore 64 y la MSX.

Figura 6: Spectrum ZX80 y monitor

En los ochenta, se creó una revolución impulsada por las empresas japonesas Sega y Nintendo, con el desarrollo de las primeras consolas de bolsillo, como la famosa Gameboy(1989) de Nintendo [Figura 7].

Figura 7: Gameboy

A partir de los años noventa siguió la progresión y complejidad de los videojuegos, apoyado en el desarrollo tecnológico gracias a la aparición de la tecnología de 16 bit y de otros progresos tanto de hardware como de software, así como los procedimientos de computación y la expansión del mundo digital. Aparecen nuevas consolas con la entrada de empresas potentes como Sony, con la Playstation en 1994 y posteriores y Microsoft con la Xbox en sus distintas versiones.

A partir de 2009, los dispositivos móviles se convirtieron en nuevas plataformas subsidiarias de videojuegos, con aparición de múltiples novedades ad hoc. Se incrementaron la utilización de juegos online multijugador y los ordenadores personales recuperaron su papel en los juegos, gracias al incremento de sus capacidades en tarjeta gráfica y computación, representando hoy en día una de las plataformas más usadas.

En relación al género de videojuego que se ha desarrollado en este trabajo, en mismo año 2009 apareció uno de los tower defense más influyentes de la historia, Plants vs Zombies (Plantas contra Zombis) [10].

Plants vs Zombies es un juego desarrollado por PopCap que consiguió vender en IOs más de trescientas mil copias en los primeros nueve días tras salir al mercado (colocándose en el primer puesto en ventas de esa plataforma) [11]. Incluso fue nominado a mejor juego casual de 2009 por la Academia de las Artes y las Ciencias Interactivas [12]. Este hecho difundió los juegos del género *tower defense* y la creación de juegos como Pixel Junk Monster, muy popular en consolas [13] y Kingdom Rush.

Kingdom Rush se acerca más a Door Defender en relación con el mercado al que se ofrece, ya que Kingdom Rush triunfó en Armor Games, una página muy similar a *minijuegos.com* en donde, todavía hoy en día, sigue siendo el segundo juego más jugado de esa página [14].

Sin embargo, Kingdom Rush no tiene ningún personaje jugable, pero sí el juego Pixel Junk Monster, aunque con la característica de que solo puede moverse por el escenario y no tiene la capacidad de atacar a los enemigos.

Por otro lado, Sanctum es un juego que sí pertenece al género *tower defense* y además dispone de un personaje con el que disparar y ayudar a las torretas, producido por Coffee Stain Studios [15]. Es el primer juego en permitir al jugador participar activamente en la defensa [16]. No obstante, este juego está desarrollado en 3D y, por ello, en muchas ocasiones es difícil localizar los puntos de llegada de los enemigos y la posición en el mapa del jugador respecto al nexo que hay que defender.

La idea más similar a la desarrollada en Door Defender es un minijuego de Terraria que surgió de la colaboración de su equipo con otro famoso *tower defense* llamado Dugeon Defenders 2 [Figura 8][17].

Figura 8: Pantallazo de Terraria del minijuego de colaboración con Dungeon Defenders 2

Como se puede observar, es en 2D y el jugador puede destruir a los enemigos.

El aspecto innovador de Door Defender, respecto a este modo en Terraria, es que no requiere elementos externos al minijuego para poder jugar los niveles siguientes: el jugador posee habilidades específicas para proteger lo que se debe defender y con la eliminación de enemigos de esta modalidad el jugador puede mejorar sus armas. En adición a esta característica, otra ventaja de Door Defender consiste en que el juego se puede jugar en web sin requerir ninguna instalación.

En resumen, Door Defender ofrece un *tower defense* con un nivel de acción muy superior a los juegos mencionados anteriormente, con unas mecánicas centradas gracias a la supresión de elementos externos o innecesarios en la jugabilidad del mismo, sin perder el elemento estratégico, ya que el empleo adecuado de las armas o la mejora de estas son elecciones que, tomadas de forma incorrecta, puedan resultar en la derrota del jugador.

Hoy en día existe mucho interés sobre este fenómeno del videojuego, como consecuencia de su implantación en las sociedades desarrolladas y su enorme proyección en el futuro, por lo que se ha llevado a cabo múltiples estudios y artículos de investigación en los que mencionan sus posibles bondades y sus efectos negativos, siendo la ludología una de las disciplinas que ha abierto un campo muy interesante para evaluar este campo del entretenimiento y de la cultura aunando ciencias sociales con la informática [18]. Como en otras actividades humanas, se han descrito también problemas de adicción, por lo que ya ha sido catalogado por la OMS como trastorno mental que puede requerir estudio y tratamiento, entrando en vigor en la nueva clasificación de enfermedades a partir de 2022 [19].

Sin embargo, con un uso correcto, se ha afirmado que los videojuegos pueden representar un elemento importante para la educación de los jóvenes y mayores tanto en aspecto cognitivos como instrumentales [20], consiguiendo la reducción del estrés, el aumento de la memoria y la conciencia estratégica, la mejora de la sociabilidad e incluso también se pueden comprobar beneficios en la salud, como por ejemplo la reducción del dolor como la mejora de la capacidad visual [21]. Además, se ha comprobado que puede ayudar a la obtención de habilidades y destreza por los cirujanos [22].

3.1 Unity 3D

Unity es un motor de renderizado en tiempo real, compatible con multitud de plataformas: iOS, Android, Windows, Mac OS, Linux, WebGL, la mayoría de las consolas de esta generación e incluso plataformas de realidad virtual. [10]

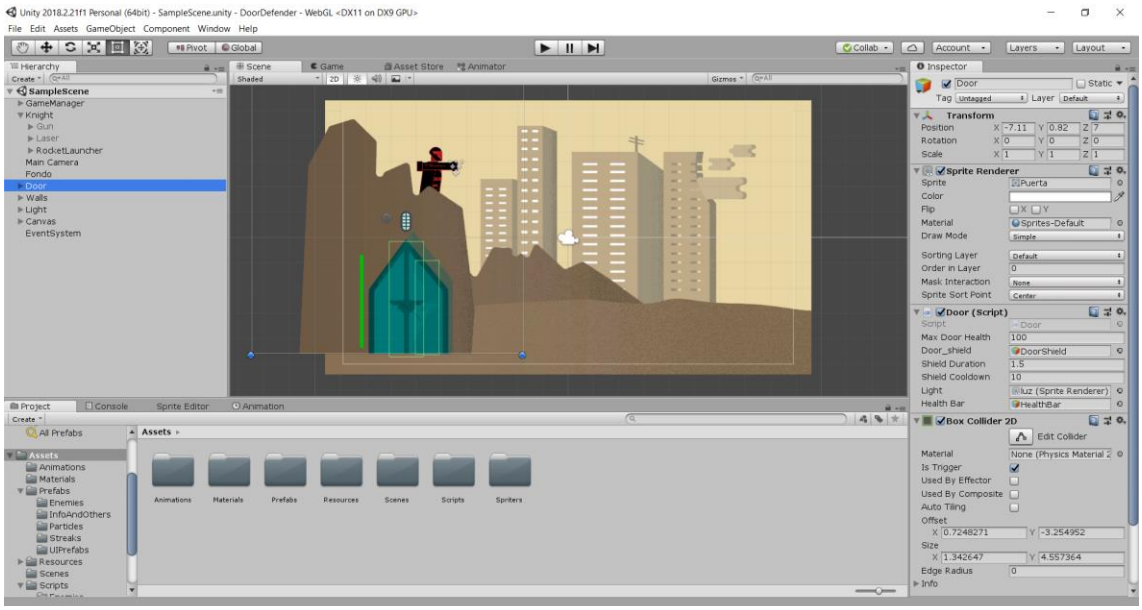


Figura 9: Entorno del editor de Unity

Además, Unity ofrece un editor con un entorno sencillo e intuitivo [Figura 9] para elaborar videojuegos, disponible tanto en Windows como en Mac Os y en Linux. El entorno permite probar el juego en cualquier momento y ofrece un diseño a medida, dependiendo si el juego a desarrollar es 3D o 2D. Asimismo, ofrece la posibilidad personalizar de forma completa el entorno e incluso añadir herramientas y extensiones nuevas desde su Asset Store.

Cualquier elemento en Unity es un *Game Object,* que se podría definir como recipiente de propiedades, que en Unity se denominan componentes. Por ejemplo, si a un *Game Object* se le adhiere el componente luz, emitirá luz en un radio determinado por los valores de ese componente [11].

Los *Game Object* se colocan en escenas. Cada escena en Unity tendrá un conjunto de *Game Object*, y se podrá transaccionar de una escena a otra dentro del propio juego a placer por el usuario.[12]

Un elemento muy importante de Unity son los *prefabs*. Son plantillas de *Game Object* creadas por el usuario con sus componentes concretos. Si el usuario utiliza este *prefab* en diferentes escenas y desea modificarlo en todas las situaciones en que aparece, puede cambiar el *prefab* y así modificar al instante la totalidad de los otros *Game Object* creados a través de este *prefab*. Al mismo tiempo, si hay varios *Game Objects* en una misma escena, pero son todos derivados de un *prefab*, consumirán muchos menos recursos que si son *Game Objects* independientes. [23]

3.1 Ventanas del Editor

Unity tiene multitud de ventanas para poder modificar los atributos y características de nuestros *Game Object*, pero en este apartado solo mencionará aquellas que han sido utilizadas en la realización del proyecto [24].

Project
La ventana *Project* (proyecto) es un explorador de archivos de nuestro proyecto de videojuego. La carpeta raíz es *Assets* y cualquier archivo que se añada a esa carpeta se incluirá en este visor para poder utilizarlo.

También existe la posibilidad de arrastrar directamente los archivos al editor. Las animaciones, fuentes de texto, materiales, *prefabs*, escenas, Scripts, imágenes, sonidos y cualquier otro recurso que utilice nuestro juego tendrá que existir aquí.

Hierachy
La ventana *Hierarchy* (jerarquía) muestra los *Game Objects* activos en la escena abierta. Los *Game Objects* hijos se mostrarán justo debajo del padre con una sangría, y podremos compactar su visualización para que solo se muestre *el Game Object* padre como representante. Si el *Game Object* es un *prefab*, se mostrará con el texto en azul.

Inspector

El inspector es una ventana que permite modificar los componentes del *Game Object* seleccionado y modificar sus atributos. También tiene la posibilidad de desactivar componentes concretos (sin borrarlos, para su uso posterior) o desactivar el propio *Game Object*. Al mismo tiempo, se puede añadir nuevos componentes o copiar los atributos de uno de ellos para transferirlo a otro *Game Object*. Cuando se modifique un *animator*, también mostrará el atributo de cada estado.

Console

En la ventana *Console* (Consola) se mostrarán todos los errores y avisos. En adición, cualquier línea de comando “Debug.Log()” que se ejecute también saldrá por esta ventana.

Scene

La ventana de escena, *Scene*, muestra los *Game Objects* activos que intervendrán en el juego. En esta ventana se podrá modificar la posición, orientación y tamaño de los *Game Objects*. Aunque estas acciones ya se podían realizar desde el inspector, desde esta ventana el proceso resulta mucho más intuitivo.

Sprite Editor

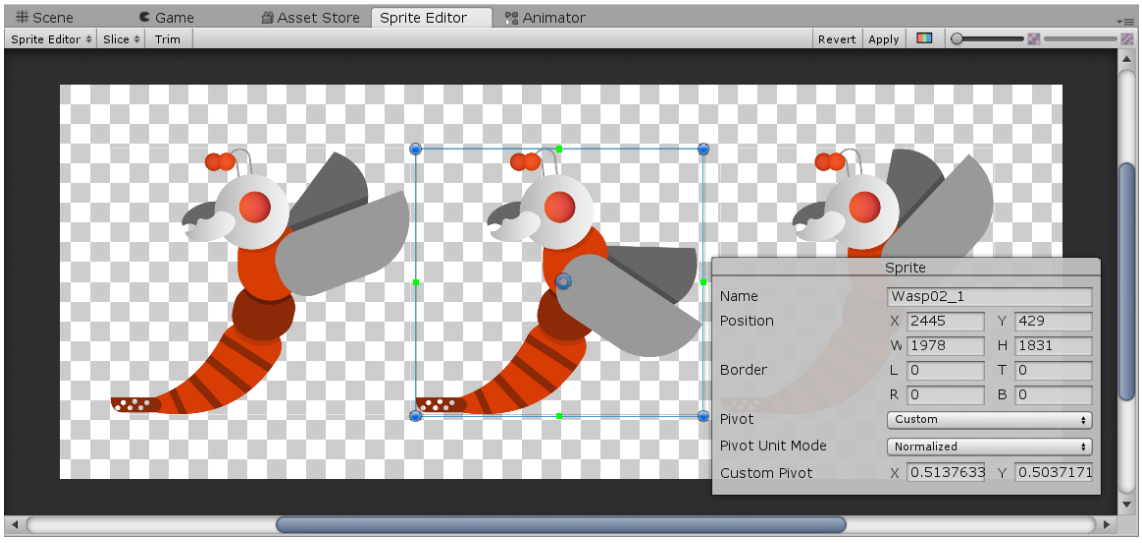


Figura 10: Slice (partición) del Spriter del enemigo Wasp

La ventana Sprite Editor permite delimitar los bordes de un *Spriter*, pero la característica más importante para la realización de este proyecto es que, dado un *Sprite* de animación como en la [Figura 10], consigue dividir sus *frames* para poder animarlo en el juego.

Game

La ventana *Game* es una previsualización del juego. Cuando se active el botón *Play* en la parte superior, se podrá jugar a lo que ya se haya desarrollado de juego. Además, se logrará poner pausa y pasar a la ventana de escena para comprobar o modificar temporalmente los atributos de un *Game Object*. Cuando se deje de ejecutar el juego, todos los atributos volverán al estado anterior al pulsar el botón *Play*.

Animation

La ventana *Animation* nos da la posibilidad de crear animaciones para los *Game Objects*. La animación en Unity consiste en una variación de los atributos con el tiempo, normalmente de forma cíclica. En los juegos 3D se suelen modificar los atributos de

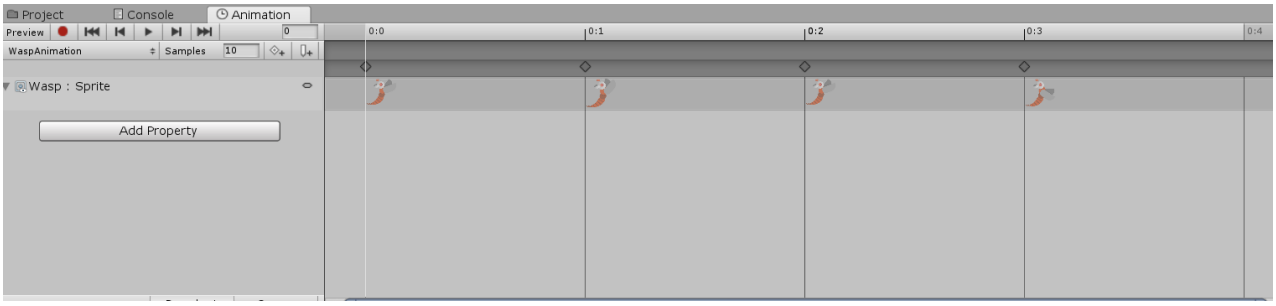


Figura 11: Ventana Animation del enemigo Wasp

posición, rotación y tamaño, pero en los juegos 2D casi siempre es el atributo *Sprite* [Figura 11].

Animator

La ventana *Animator* nos permite configurar qué animación se encontrará activa dependiendo o bien del tiempo transcurrido o bien de los atributos que pueden ser modificados por los Scripts. También podemos añadir transiciones y nuevos estados [Figura 12].

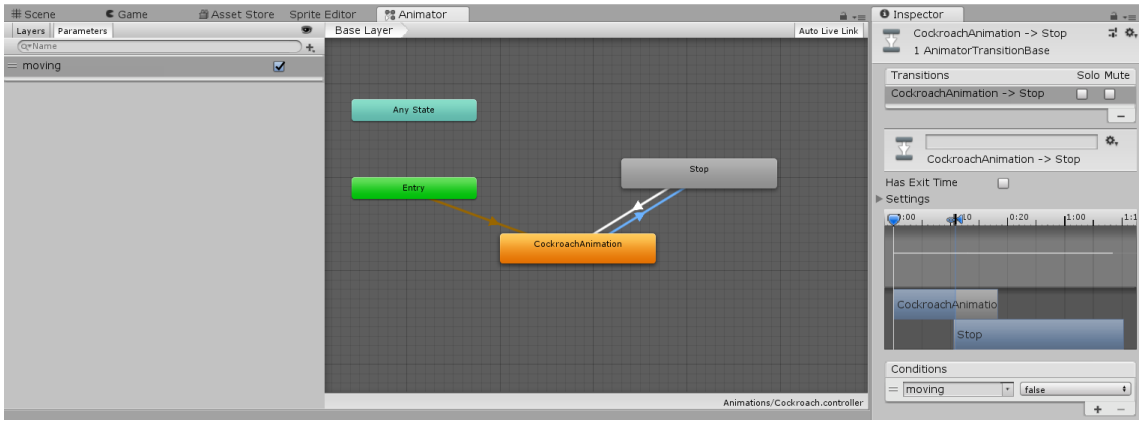


Figura 12: Grafo de estados de animación del enemigo Cockroach.

3.2 Componentes de Unity

Los componentes de Unity dotan de características a los *Game Object* del proyecto. Cada componente tiene unos atributos que pueden ser modificados tanto por el inspector como por código usando *Scripts*, que es un componente más.

En este apartado no se comentará todos los componentes de Unity, solo aquéllos que han tenido un papel relevante para la implementación del proyecto [25].

Transform

El componente *Transform* determina la posición, tamaño y rotación de un *Game Object*.

Todos los *Game Objects* tienen este componente, aunque aplique en el editor a crear un *Game Objects* vacío. Los objetos pertenecientes a la interfaz tienen una variante llamada *Rect Tranform*, que ofrece más atributos de personalización.

Sprite Renderer
En los proyectos Unity en 2D la mayoría de los elementos del juego posee este componente. Determina el aspecto que tendrá el *Game Object*, su color (el color es una capa por encima de la imagen original, si es blanca será exactamente igual que el *Sprite* asociado) y transparencia.

El color tiene cuatro componentes, los cuales se representan con un *float* que tiene un espectro de 0 a 1. Esas cuatro componentes son nivel de rojo, nivel de verde, nivel de azul y nivel Alpha. El nivel Alpha se determina como transparente, es un *Spriter*. Si es 0 es transparente y si es 1 totalmente opaco.

Button
La componente *Button* tiene un apartado “On Click()” que permite añadir otros *Game Objects* con *Script* (o a sí mismo) con una función sin argumentos. Una vez el juego en ejecución, si el jugador pulsa sobre el botón se ejecutará dicha función.

Collider
Un *Collider* (en este proyecto *Collider* 2D) es el que da forma “tangible” a un objeto en Unity. También se pude utilizar para detectar otros objetos como un área (activado “Is Trigger”) para posteriormente ejecutar un bloque de código si el *Game Object* posee un componente de tipo Script.

Rigidbody
Permite al motor de físicas actuar sobre el objeto al que se le aplica el componente. Los *Colliders* solo detectarán *Game Objects* que posean este componente.

Animator
Permite a un *Game Object* tener animación.

Particle System
El componente *Particle System* te permite que el *Game Object* emita pequeñas partículas. Se utiliza para efectos visuales mayoritariamente y da sensación de dinamismo y fluidez al juego.

Script
Te permite asociar un código a un *Game Object*. Cualquier atributo público que tenga un *Script* podrá ser modificado desde el editor e incluso poder arrastrar de la ventana de proyecto o la de jerarquía directamente a la casilla del inspector.

La mayoría de los Scripts heredan de MonoBehaviour.

3.3 La clase MonoBehaviour

MonoBehaviour tiene dos métodos principales: *Start* y *Update*.

Start se ejecuta cuando el *Game Object* se crea, y constituye el método donde se suele inicializar los atributos del *Script*.

Update es un método que ejecuta cada *frame* (imagen) del juego. Por ejemplo, si en *Update* hay un código que permite a un enemigo avanzar una unidad hacia adelante y el juego se desarrolla a 60 imágenes por segundo, al pasar dos segundos el enemigo habrá avanzado 120 unidades. El problema surge porque no se puede determinar a cuántas imágenes por segundo funcionará el juego por lo que, dependiendo de la capacidad computacional del sistema que ejecute un juego, este tendrá comportamientos distintos. Para solucionar este problema, existe "*Time.deltaTime*" que mide el tiempo transcurrido desde el último *frame*. Así pues, si multiplicamos la distancia que se decide que debe recorrer el enemigo por "*Time.deltaTime*", el enemigo se moverá a la misma velocidad independientemente del sistema que ejecute nuestro juego [25].

4. Análisis y Diseño

Antes de comenzar el proyecto, los objetivos planteados se centraron en la creación de tres enemigos diferentes, tres armas y la posibilidad de mejorarlas.

Los primeros enemigos concebidos eran unos escarabajos con una ametralladora en la espalda, que andaban lentamente y con nivel de vida. El segundo tipo de enemigos lo constituía un pequeño robot de velocidad alta que cuando estuviese cerca de la puerta explotase. El tercer enemigo se caracterizaba por ser un adversario con la capacidad de crear a los otros dos.

Las armas elegidas para su utilización por el jugador serían una pistola, un arma láser y un cañón cuya munición (bolas sólidas de peso elevado) tuviera trayectoria descendente para impactar en un área. Cada una de estas armas podría ser susceptible de mejoras en cualquier momento, mediante un menú emergente. Se utilizaría cada tipo de armamento dependiendo de las características del adversario. Para el enemigo básico, la pistola; el arma láser para los rápidos y el cañón para el invocador.

Al principio, la estructura original iba constar de un *script* para cada una de las armas. También se decidió la asignación de un *Script Manager* para gestionar los otros *managers* que en un principio se iba a incluir: un *manager* para el nivel encargado de invocar a los enemigos, un *manager* para el inventario, que gestionaría la chatarra obtenida y un *manager* para las mejoras, que dispondría de una lista con los materiales y con anotación de las necesidades para cada uno. En adición, se utilizaría un *Script* diferente para cada tipo de enemigo y un Script que gestionase la vida de la puerta y el fin del juego.

La planificación se realizó en tres partes:

- la primera parte centrada en que el juego resultase satisfactorio de jugar, con gráficos de la mínima expresión.
- Una segunda fase, con los gráficos ya establecidos donde implementamos las colisiones, los enemigos, la puerta a defender, una interfaz básica y un primer nivel con mejoras de arma sencillas.
- Por último, una tercera fase donde la interfaz final ya estaba decidida, los niveles y enemigos pendiente de su desarrollo y un enemigo final para terminar el juego.

El orden de implementación de la planificación fueron los siguientes hitos:

1. **Estructura del proyecto.**
2. **Fase uno.**
   2.1. Apuntado.
   2.2. Disparo de pistola.
   2.3. Disparo de arma láser.
   2.4. Cambio de arma.
   2.5. Disparo de cañón.
3. **Fase dos**
   3.1. Enemigo básico.
   3.2. Puerta.
   3.3. Habilidades de la puerta.
   3.4. Obtención de objetos.
   3.5. Mejora de armas.
   3.6. Menú principal.
   3.7. Pantalla de Fin del Juego.
4. **Fase tres**
   4.1. Enemigos complejos.
   4.2. Mejoras avanzadas.
   4.3. Interfaces finales
   4.4. Niveles**.**
   4.5. Jefe final.
5. **Testeo y revisión de errores.**

4.1 Diseño de mecánicas

El control del personaje se realizará mediante el ratón. Moverlo por la pantalla servirá para apuntar, los disparos se ejecutarán con el botón izquierdo del ratón y con el botón derecho la habilidad del escudo. Con la ruleta del ratón se cambiará de arma.

La cámara estará fija en el escenario. Se podrá observar la puerta y por la derecha irán apareciendo los enemigos.

La destrucción de enemigos hará que pierdan materiales que servirán para mejorar las armas que ya posee el jugador.

Los tres tipos de enemigos que habrá en el juego serán:

- Un enemigo muy básico.
- Un enemigo volador.
- Un enemigo muy resistente pero lento.

Las mejoras de arma se realizarán entre niveles, y se podrá elegir un nivel anterior o repetir el que se acaba de jugar.

Los niveles constarán de tres fases:

- Primera fase: se presenta un nuevo enemigo en solitario o con enemigos muy débiles,
- Segunda fase: se integra a ese enemigo con enemigos más duros de otras fases.
- Tercera fase: se presenta una versión poderosa de ese enemigo acompañado de la versión normal de dicho adversario y otros enemigos poderosos de fases anteriores.

El último nivel no seguirá este esquema, sino que exhibirá una gran horda con todos los enemigos presentados y culminará con la aparición de un enemigo muy poderoso que pondrá a prueba todas las armas del jugador.

4.2 Elementos modificados en la implementación

A pesar de haber establecido este esquema de diseño del juego, hubo que modificar muchos elementos durante la implementación. Cuando se estaba desarrollando el cañón, se observó que su manejo no producía el resultado esperado. Por esta razón, se decidió idear un lanzacohetes, el cual necesitaría una clase adecuada para el propio misil disparado. Además, la disponibilidad de un *Script* para cada arma resultó no tener mucho sentido, ya que la lógica resultaba muy similar. Por este motivo, se unificaron en un *Script* llamado “Knight” que, dependiendo del arma equipada con un *switch*, se comportaría de un modo u otro.

Después de la **fase uno**, contactamos con el colaborador que iba a diseñar los gráficos del juego. Al comentar la idea global y desarrollo del juego, con la descripción de las criaturas mitad artrópodo y mitad máquina, se decidió que estaría mejor modificar también otro elemento de la idea original: al comprobar las características del espacio vacío existente en la prueba del arma, se planteó la idea de que uno de los enemigos pudiera volar.

Al diseñador gráfico se le remitió los bocetos de las armas, propias de un enemigo muy básico. También se le encargó el diseño del personaje principal, de las tres armas y de dos enemigos. Cuando tuvo listo los diseños, se decidió la creación de un tercer enemigo, cuya razón de ser fue sustituir al invocador, ya que este último constituía un elemento que no terminaba de ajustarse dentro del diseño del juego.

En la implementación de la **fase dos** también se modificaron objetivos Todos los enemigos compartirían el mismo *script*, ya que sus acciones serían principalmente las mismas. La utilización de un arma específica para cada tipo de enemigo resultaba ciertamente tediosa, así que se concluyó que el uso de cada arma debería depender del posicionamiento de los enemigos, no de sus características. Además, la realización de mejoras en mitad de un nivel resultaba anticlimático, por lo que se decidió que solo se podrán realizar después de finalizar un nivel, cambiando la idea original de su concepción.

El *manager* de las mejoras se fusionó con el del inventario. Para realizar las mejoras se requiere acceder al inventario, y resultaría más sencillo un solo *manager* que se encargara de las dos funciones. En este proceso se vio también la conveniencia de crear un nuevo *manager* cuya función sería gestionar las interfaces.

Finalmente, se resolvió una última modificación: eliminar la idea de componer un menú principal. El juego es muy sencillo, y situar un menú con solo dos opciones antes de jugar en realidad entorpecía la experiencia. Por lo tanto, se sustituyó el menú por un mensaje al principio del juego, mensaje cuyo objetivo es informar sobre los creadores del juego y los diseñadores de los gráficos.

5. Implementación

La implementación puso a prueba muchos de los conocimientos cursados durante toda la carrera. La facilidad para encontrar soluciones o averiguar cómo desarrollar la programación de cada elemento para que realice lo que se desea, no hubiera sido posible sin el entrenamiento en las prácticas de las distintas disciplinas y, sobre todo, las de la rama de computación. El lenguaje utilizado, C#, solo fue utilizado en ingeniería del software, pero todo el conocimiento adquirido en java o en Python no resultó infructuoso ya que estos tres lenguajes tienen similitudes entre sí. Además, si se consigue alcanzar la capacidad de diseñar algoritmos, pierde relevancia el tipo de lenguaje utilizado.

El entorno de programación utilizado fue Microsoft Visual Studio, la versión de Unity utilizada fue Unity 2018.2.21f1 y para ajustar algunas de las imágenes se utilizó Paint.net.

La implementación ha sido desarrollada en tres fases temporales. La primera fase a finales de enero, la segunda fase en la última quincena de abril y la última fase durante la segunda mitad de junio.

Al desarrollar el proyecto utilizando Unity, la estructura suele ser similar en la mayoría de los juegos al inicio del proceso creativo. La organización de la carpeta *Assets* se llevó a cabo mediante una carpeta *Prefabs*, una carpeta *Scenes*, otra Scripts y finalmente una *Spriters*, donde se irán ubicando los diseños que mi colaborador realizará. Como aún no disponía de los *Spriters*, la primera escena fue delineada con una técnica denominada *White Boxing*, que consiste en representar los elementos del juego como cajas blancas simples con la idea de probar la jugabilidad y, si resultara interesante, ya sustituirla posteriormente por modelos más complejos.

5.1 Knight.cs

El primer *script* que se programó fue “Gun” (pistola), que pasaría a ser “Knight” por lo que se comentará más adelante.

La primera acción que desarrollar fue la respuesta de las armas al movimiento del ratón, es decir, conseguir apuntar con ellas con la mayor precisión posible. Para ello, en el método *Update* se escribió un algoritmo que toma la posición del ratón, utiliza la cámara para saber su situación relativa respecto a ella y se obtiene el ángulo. Después, no se permite que ese ángulo sea mayor que 70 o menor que -70f, para que el personaje no gire el brazo de una forma imposible. Más adelante, se obtiene su *Quaternion*, que consiste en una forma compacta que tiene Unity para representar las rotaciones [31]. Por último, le aplicamos al *spriter* del arma la rotación, multiplicándola por

“Time.deltaTime” para que su velocidad de giro no dependa del sistema que ejecuta el juego.

Para el disparo, antes del algoritmo anterior, dentro del *Update*, con un “Input.GetMouseButton(0)” determinamos si el botón izquierdo del ratón ha sido pulsado. Si se ha dado el caso, crearemos instancia de una bala.

La bala es un *prefab* con un *script* que en su *Update* tiene que avanzar hacia adelante. Como cuando se instancia también se pasa la rotación del brazo, la bala saldrá alineada con el arma, y así se moverá a la dirección apuntada.

Por otro lado, el arma láser crea un haz de luz roja en vez de una bala, haz luminoso que atraviesa la pantalla desde el cañón del arma hasta la posición del ratón en el momento de disparar [Figura 13].

Figura 13: Arma láser en acción

Para obtener una mejor sensación en el uso de esta arma, cuando se dispara el láser la rotación se quedará fija durante toda su duración. No se podrá realizar movimiento con el brazo, por lo que el jugador tendrá que pensar muy bien dónde disparar el arma, ya que se fijará en esa posición unos segundos.

El cambio de arma se realiza girando la rueda del ratón. “Input.GetAxis (‘Mouse ScrollWheel’)’” devuelve un numero positivo o negativo dependiendo de la manera en la que se gira la rueda. Cuando el giro se produzca, el *game object* de un arma pasará a estar activo y el que está activo se desactivará. En definitiva, este diseño permite que las tres armas estén situadas en el mismo punto, pero solo una quedará activa.

El lanzacohetes es muy similar a la pistola, solo que, en vez de una bala, lanzará un *prefab* cohete.

Para optimizar el juego, en los bordes de la pantalla, pero fuera de la vista del jugador, se establecieron *collider* 2D cuya misión es eliminar cualquier proyectil que penetre en ellos, para que las balas perdidas dejen de existir. Para que este hecho pudiera suceder, las balas y cohetes debían tener un *rigidbody* y un *collide*r 2D. Y el *collider* de los muros debía ser *triggers*. Los muros tendrían un pequeño *Script* para que cada elemento que entre en su *collider* sea destruido. De este modo, cualquier proyectil que sale del límite de la pantalla será eliminado y no consumirá recursos.

El suelo funcionaría de forma similar. Por esta razón, surgió el problema de la desaparición de las balas de forma repentina al impactar con el suelo y esta perspectiva ofrecía una impresión muy artificial. Para solucionar el problema, se creó un objeto vacío con *particle system*. Este objeto vacío se instanció como *prefab*, y cada vez que una bala impactaba en el suelo, se instanciaba el *prefab* de la partícula. Este *prefab* tiene un sencillo *script*, que consiste en que después de una existencia de un segundo, se autodestruye. De esta manera, el *prefab* de partículas se crea, reproduce su efecto de partículas una vez y se autoelimina para liberar espacio.

Para la implementación del cohete se creó otra partícula, pero con el mismo s*cript*, pero en vez de simular un impacto en la tierra, se diseñó que se produjera una explosión al contactar con esta superficie [Figura 14].

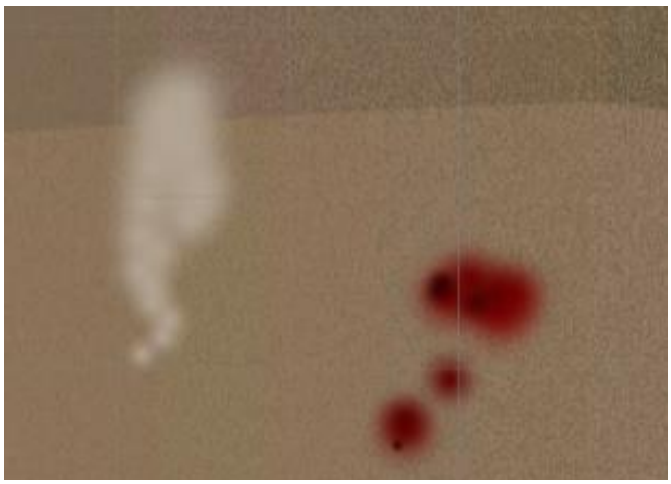


Figura 14: Partículas de impacto en el suelo y de explosión del cohete

Las armas tienen los siguientes atributos que pueden ser mejorados a lo largo del juego:

- Común en las tres:
  - Velocidad de disparo.
  - Velocidad de apuntado.
- Pistola:
  - Daño por balas.
- Laser:
  - Daño por *tick* del láser.
  - Duración del láser (este atributo no cambia).

- Lanzacohetes:
    - Daño por los cohetes.
    - Daño de explosión de los cohetes.

5.2 Enemy.cs

Para poder realizar la prueba de las armas, se creó un *Game Object* con un *collider* 2D. El *script* de Enemy tiene un atributo público denominado vida, y cada vez que entra una bala o un cohete, ese atributo disminuye y si se alcanza el punto cero o inferior, el enemigo se borrará de la escena. Para indicar al jugador que su ataque está produciendo el efecto esperado al recibir un proyectil, el aspecto del enemigo será un poco más oscuro durante unos segundos. Este cambio de coloración se realiza mediante la obtención del componente *Sprite Render* y la modificación de su atributo *color* y, tras unos segundos, volverá a su color original [Figura 15]. También se desplazará ligeramente hacia atrás.

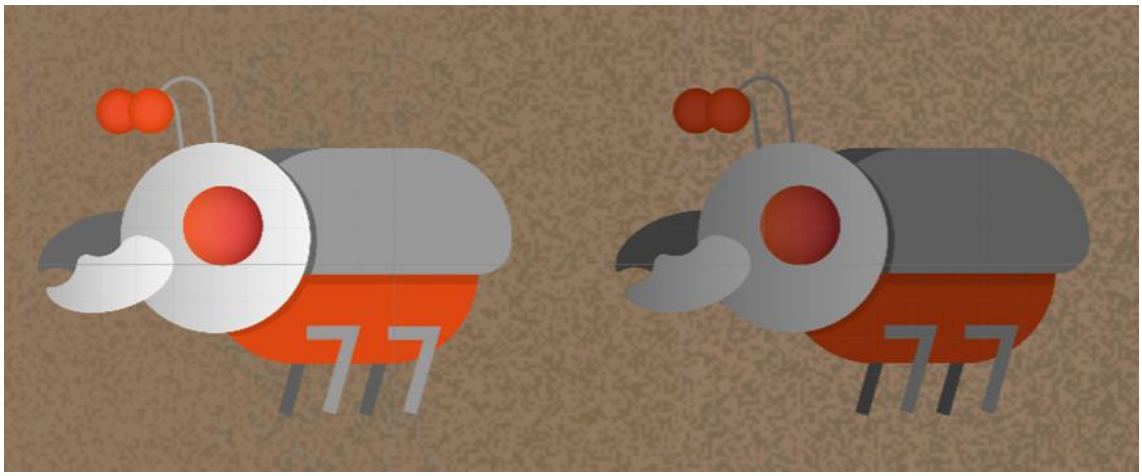


Figura 15: A la izquierda, enemigo en estado por defecto, a la derecha, al recibir daño

Un problema surgió con el arma láser. Al efectuar contacto continuo con dicha arma, el enemigo era fulminado demasiado rápido. Para solucionar este problema, el tiempo en el que el enemigo permanece con la coloración gris, es invulnerable. El escaso tiempo que permanece en gris es suficiente para que el láser produzca el daño esperado y, además, solo para el láser se desactivó el efecto de empuje, ya que el efecto resultaba extraño (al producirse muchos empujones en poco tiempo).

El daño que produce el arma láser funciona de forma distinta, dado que no es un único impacto, sino que origina un daño continuo. El láser producirá daño una vez cada diez milisegundos, no hará retroceder al enemigo.

La implementación del ataque fue sencilla. El enemigo avanza y cuando se encuentra a una distancia cercana de la puerta, el defensor dispara contra el enemigo.

En esta implementación básica de ataque, se mejoró el *script* de la forma siguiente: al detectar un proyectil, se discrimina si es una bala, un láser o un cohete. Según sea el tipo, se accede al daño que produce ese determinado proyectil y ese valor es el que se resta a la vida del enemigo.

Para el efecto de daño en área del cohete, cuando éste impacta en algún punto crea un *game object* vacío con un *collider* 2d circular. Cuando un enemigo detecta ese *game object*, sufre un daño por explosión (un valor predeterminado instanciado como variable pública).

5.3 Door.cs

La Puerta tiene un *collider* para detectar las balas de los enemigos. Cada vez que recibe un impacto de esta munición, comprueba su atributo de daño y se lo resta a la salud. Si la salud de la puerta es inferior o igual a cero, se cambia de escena a la de fin del juego. La habilidad del escudo de la puerta se ejecuta con el botón derecho ("Input.GetMouseButton(1)"). Al pulsarlo, se crea una instancia de una versión de la puerta azulada que destruye cualquier proyectil que lo toca (esto se realiza con un *collider* y un *script* muy similar a los explicados anteriormente).

El escudo tiene un tiempo de recarga y una duración. Para controlar ese tiempo se dispone de tres variables. Dos variables para determinar el tiempo que debe estar activo el escudo y la tercera para establecer la cantidad de tiempo que tarda en recargarse. Es decir, la última variable es un contador para conocer la cantidad de tiempo que lleva el escudo activo o el tiempo que lleva recargándose.

El proceso es el siguiente:

> Se activa el escudo, la variable Contador se instancia a cero y se le suma "Time.deltaTime" en el *Update*. Cuando esa variable sea mayor o igual a la variable "DuracionEscudo" se desactiva el escudo, se vuelve colocar la variable Contador a cero y se repite el mismo proceso, pero esta vez con "DuracionRecarga". Cuando Contador sea cero, volvemos a dejar que el jugador pueda usar el escudo (esto se controla con un *booleano*).
>
> Para mostrar al jugador el estado del escudo, lo indicará una luz situada arriba de la parte superior de la puerta. Cuando la luz sea azul, el escudo se podrá utilizar. Al manejar el escudo, la luz irá transaccionando del azul al rojo, y cuando sea completamente roja, el escudo estará a punto de desactivarse. Al llegar a esta situación, la luz de arriba de la puerta se apagará y lentamente irá tornando a verde, indicando de esta forma el tiempo restante para que el escudo vuelva a estar disponible. La luz retornará a su estado original de azul, revelando que el escudo está listo para su reutilización. Para conseguir este efecto, la luz consta de dos capas: la carcasa de la luz y una imagen de fondo. La imagen de fondo es la que hay que modificar para conseguir el efecto deseado. Cuando Contador es modificada en sus respectivos bloques, asignaremos el nuevo color al *Sprite Render* de la imagen de fondo.
>
> En la situación de escudo activado, el nivel de azul será "1f - Contador/DuracionEscudo" (f significa que el número es de tipo *float*) y su componente roja será "Contador/Escudo". Así pues, cuando se encuentre la energía del escudo al cincuenta por ciento de duración, tanto el nivel de rojo como el de azul se encontrarán a la mitad (0.5f).

Para la “DuracionRecarga”, empezaremos en los tres niveles de color a 1 (el color negro) y el nivel de azul y rojo en cada iteración irá cambiando como “1f - Contador/DuracionRecarga” [Figura 16].

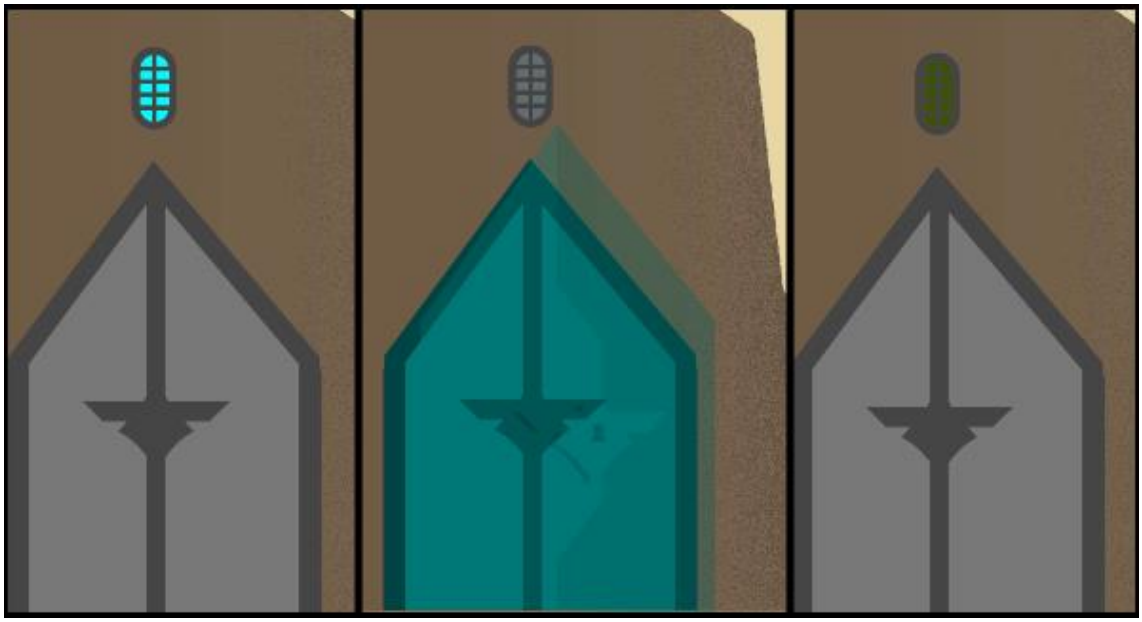


Figura 16: Comparación de Puerta con escudo listo para su uso, Puerta utilizando el escudo y Puerta recargando el escudo.

Por último, para indicar la cuantía de vida resultante de la puerta, se consigue con un *sprite* de una barra negra que es *game object* padre de un *game object* hijo con un componente *sprite* de una barra verde. El tamaño de la barra verde será “VidaActualPuerta/VidaMaximaPuerta” en el eje Y (al tratarse de la barra vertical). Además, si ese valor es menor que 0.5f, la barra pasará de verde a amarilla y si es menor a 0.25f entonces se iluminará en rojo. Cuando el tamaño de la barra sea menor que un cuarto, todos los ataques restarán la mitad de su daño original. El jugador desconocerá este hecho y le producirá la sensación de haber salido victorioso en el último suspiro de su puerta.

5.4 Managers

Los *managers* son *scripts* cuyo acceso se logra desde otro *script* en cualquier parte del código. Se encargan de llevar elementos comunes, como el inventario. Todos los *managers* son inicializados por el *manager* principal, el cual se encarga de iniciar todos sus “Startup”, que es el equivalente al “Start” de los MonoBehavior pero aplicados para los *managers*. Después de que todos los *managers* le hayan indicado al *manager* principal que han sido iniciados correctamente, éste iniciará el juego.

En este juego en concreto, además del *manager* principal, existen cuatro *managers* [Figura 17]:

- *InvetoryManager.cs*: es el encargado de gestionar la chatarra liberada por los enemigos, la cual llamaremos *ítems* a partir de ahora. También gestiona las mejoras, las necesidades para poder llevarlas a cabo y el impacto que tienen en las armas realizadas. Para gestionar el inventario, utiliza dos diccionarios, uno para los *items* y otro para las mejoras.
- *LevelManager.cs*: este *manager* determina los enemigos que van a ser invocados en el nivel, la situación de pausa del juego, y el paso a la fase de mejoras cuando el jugador ha conseguido derrotar a todos los enemigos. Para invocar a los enemigos, dispone de una referencia a todos sus *prefabs*. Cada nivel funciona de la siguiente forma:
    - El nivel consta de una serie de oleadas, grupos de enemigos. LevelManager invoca la primera oleada (tiene un contador de tiempo para no invocarlos todos a la vez).
    - Cuando en la escena no exista ningún enemigo, invoca la segunda oleada.
    - Cuando no queden más oleadas, abre el menú de mejoras y aumenta el valor de nivel actual. Este valor determina que grupos de oleadas se invocará.
- *UIManager.cs*: gestiona los elementos que dan información extra al jugador. Es el encargado de que no haya más de una interfaz abierta, que aparezca la información en pantalla cuando se obtiene un ítem y también que se muestre los ítems que sean necesarios para cada mejora. Inicialmente, este *manager* no se esbozó en la planificación, pero resultó imprescindible para la gestión eficiente de todas las interfaces.
- *KillStreak* es un *manager* especial. Fue un *manager* que surgió en el último momento para mostrar al jugador mensajes de vitoreo cuando haya conseguido eliminar varios enemigos a la vez. Tiene un peso ínfimo comparado con los otros tres, pero su presencia puede dar motivación al jugador, de una manera puramente estética.

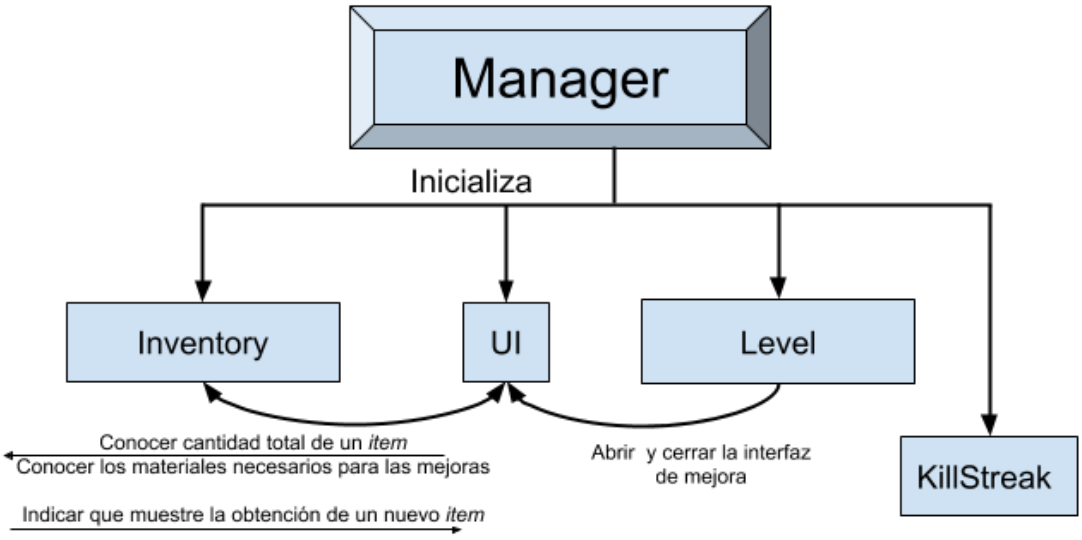


Figura 17: Esquema relacional de los *managers*

5.5 Interfaces

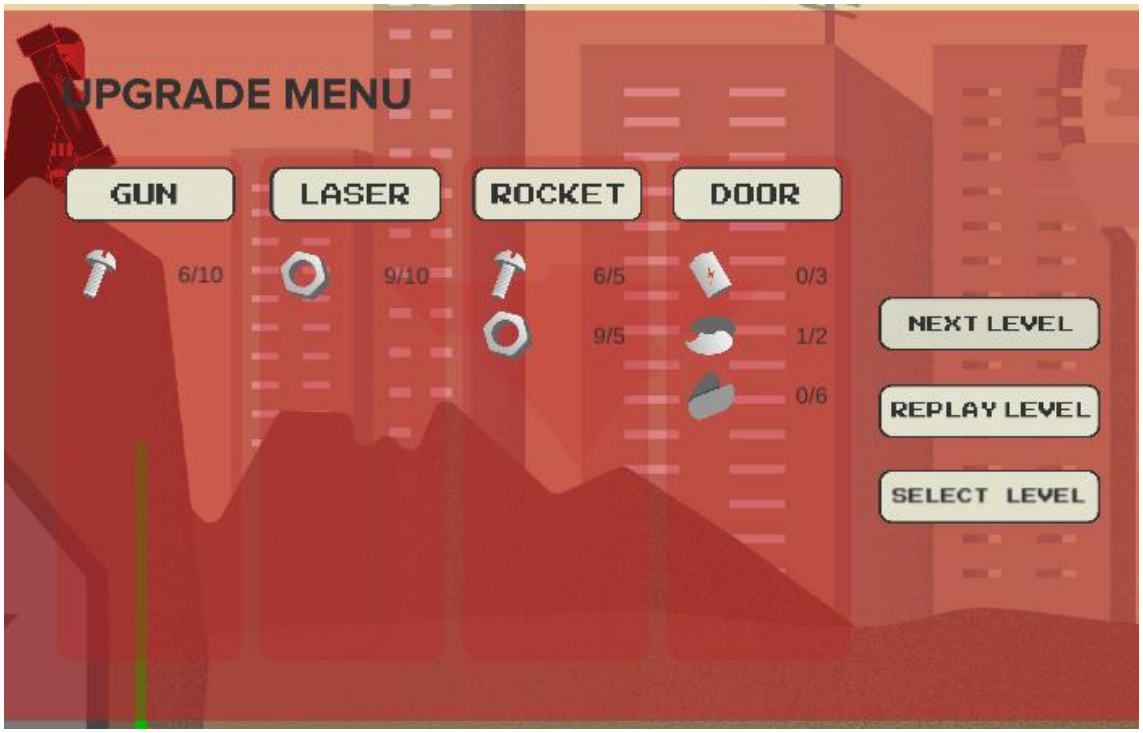


Figura 18: Interfaz de mejora dentro del propio juego.

La interfaz más importante es la encargada de mostrar los materiales necesarios para poder mejorar las armas del jugador [Figura 18]. Está interfaz se muestra tras finalizar un nivel. Permite diversas opciones:

- Mejora de las armas y de la puerta.
- Selección de un nivel anterior.
- Repetición del último jugado o de avanzar al siguiente nivel.

Otro elemento de la interfaz muy útil es un *popup* que aparece cada vez que una criatura libera chatarra [Figura 19].

Figura 19: *Popup* del *item* obtenido (en este caso se han obtenido 11 tornillos y en total se dispone de 42).

Estos aparecen a la derecha de la pantalla, con una animación de entrada, y van desapareciendo poco a poco. Cuando un enemigo muere, notifica al *manager* del

inventario para que añada el *ítem* al mismo con información que le pasa al enemigo. El *manager* de inventario comunica estos datos al *manager* de *interfaces* que instancia un *prefab* con la información transmitida. Para que el icono pueda surgir, en la carpeta Assets existe una carpeta llamada Resources, donde hay otra carpeta llamada Icons con una imagen con cada tipo de objeto. En el caso de la imagen de ejemplo, existirá un *sprite* llamado "Tornillo" y con la sentencia "Resources.Load<Sprite>('Icons/' + name);" obtendríamos el *sprite* asociado a la variable *name*.

Otras *interfaces* diseñadas, con un papel no tan relevante, son:

- Menú de información: contiene información sobre los controles y los créditos de los creadores del juego. Es lo primero que se muestra al iniciar el juego, y cuando se pulse sobre el botón que contiene se cerrará y empezará el juego.
- Menú de opciones: se muestra al pulsar sobre el botón en la esquina superior izquierda. Cuando se abre este menú, pausa el juego hasta que se pulse el botón "*Resume*". También tiene un botón "Info" que abre el menú de información.
- Menú de selección de nivel: muestra los niveles disponibles y te permite rejugar un nivel anterior al pulsar sobre su botón correspondiente.
- Fin del Juego: aunque es una escena dividida en partes, se muestra cuando la puerta pierde todos sus puntos de vida. Permite empezar la partida de nuevo.
- Juego termina: interfaz que se muestra al acabar el juego. Permite empezar una nueva partida o seleccionar un nivel anterior.

5.6 Enemigos avanzados

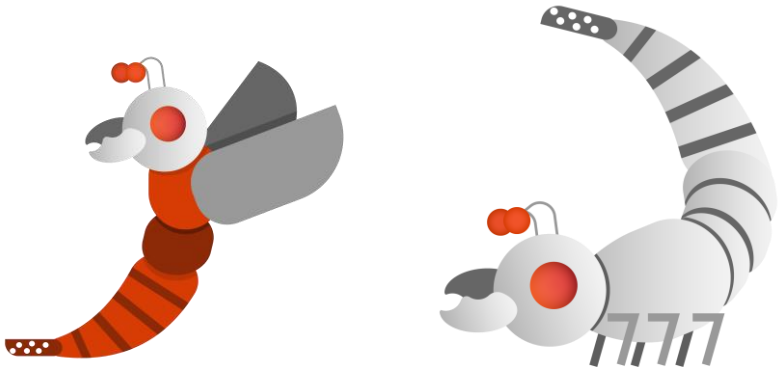


Figura 20: Enemigos. Izquierda Wasp (avispa), derecha Scorpion (escorpión).

Para poder realizar de forma sencilla distintos enemigos con un mismo *script* [Figura 20], se hicieron públicos los siguientes atributos:

- *MaxHealth*: determina la vida máxima del enemigo.
- *Speed:* determina la velocidad a la que avanza hacia la puerta.
- *ShotSpeed*: establece la rapidez de ataque del enemigo en cuestión.
- *Attack Range*: determina la distancia a la que tiene que encontrarse el enemigo para poder iniciar un ataque.
- *Damage*: el daño que hacen los proyectiles a la puerta.
- *Drops*: una lista con la chatarra que pude liberar el enemigo al morir.
- *SpawnY*: establece el punto del eje de las Y en el que el enemigo es invocado.

Además de estos atributos, algunos enemigos poseen un ataque cargado, que consiste en la emisión de una ráfaga de balas. Los enemigos que poseen está habilidad, tienen un s*cript* llamado AtaqueCargado.cs, que les permite utilizar está función [Figura 21].

Los enemigos del juego son:

Nombre	**MaxHealth**	**Speed**	**ShotSpeed**	**Damage**	**Otros atributos**
Cockroach	14	2	1’5	2	
Small Cockroach	5	2’5	2	2	
Big Cockroach	40	1’75	1	10	
Wasp	100	1’5	1’5	6	SpawnY es superior
Big Wasp	200	1’5	1’5	8	SpawnY superior y con ataque cargado
Scorpion	500	1	---	4	Solo realizan ataques cargados
Scorpion Boss	1500	0’75	---	16	Especial. Mirar apartado jefe final

La mayoría de los enemigos liberan “Tornillo”, “Tuerca” y “Pinzas”. Los jefes de nivel, como Big Cockroach y Big Wasp sueltan “Pila” y PilaG” respectivamente. Las “Ala” solo los emiten los enemigos tipo Wasp, y “Cola” los enemigos tipo Scorpion. Para determinar qué objetos puede soltar cada enemigo, disponen de un array de *strings* donde cada elemento es un tipo de chatarra posible. Cuando son destruidos, eligen un elemento aleatorio de ese *array* y una cantidad aleatoria de 0 a 3.

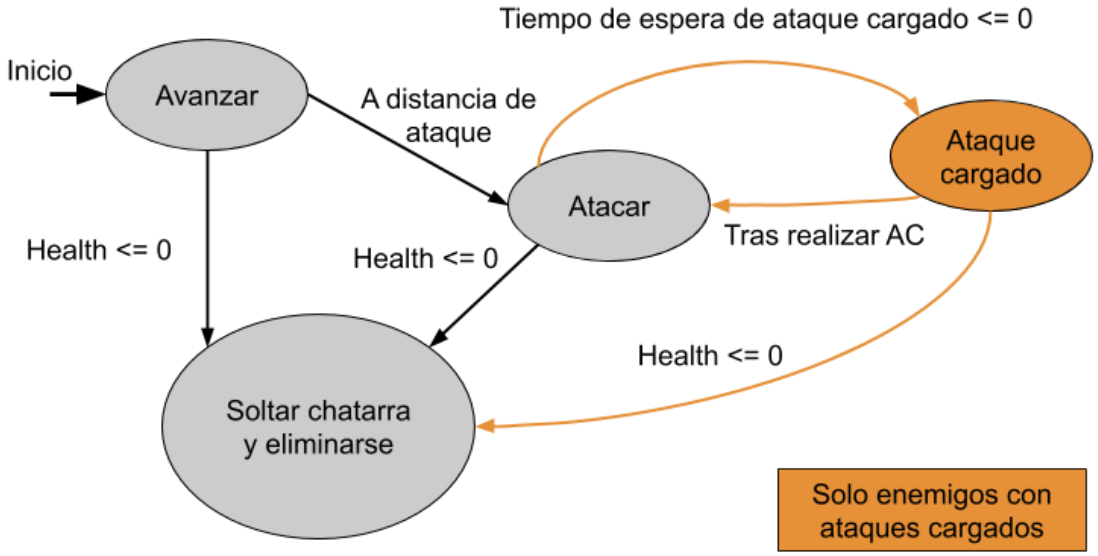


Figura 21: Lógica de los enemigos.

5.7 Pequeños detalles

A continuación, se describe algunas mejoras que no ofrecen un impacto muy grande, pero que perfeccionan la experiencia en general y fueron implementadas cuando el juego estaba prácticamente desarrollado:

- Vida de los enemigos: a los enemigos se les otorgó una barra de vida como la de la puerta, pero horizontal. Funcionan de la misma forma.
- Selección de nivel: un botón en la ventana de mejoras que permite jugar un nivel anterior para conseguir materiales. Al pulsar el botón, se abre una ventana con un botón para cada nivel que el jugador ha completado.
- KillStreak: su funcionamiento se explicó en *Manager*; se implementó cuando el juego estaba prácticamente completo.
- Los enemigos al morir crean un sistema de partículas que simula que han explotado. Al inicio del proceso de diseño del juego, simplemente desaparecían, pero este detalle añadido hizo mucho más divertido derrotarlos.
- Los ataques que anula el escudo de la puerta tienen un sistema de partículas que simula que el proyectil ha sido absorbido por el escudo. Así, el jugador es totalmente consciente de la efectividad del escudo.

5.8 Jefe final

Figura 22: Jefe final de Door Defender.

El jefe final [Figura 22] es el enemigo más duro y difícil del juego. Está planteado como una gran amenaza que pone a prueba las habilidades del jugador y solo puede ser vencido si se poseen todas las mejoras y en el momento en el que el jugador haya conseguido entender la utilización del arma adecuada, dentro de su arsenal, para cada situación de amenaza concreta.

El jefe es una versión más poderosa del Scorpion. Tiene el *script* Enemy.cs, pero utiliza un ataque personalizado. Utiliza dos s*cripts* nuevos, uno para su comportamiento y otro para un escudo con el que viene equipado, muy similar al de la puerta.

El jefe final tiene tres fases [Figura 23]:

- En la primera fase, avanza lentamente hacia la puerta, y transcurrido un tiempo determinado, activa un *game object* hijo, que relanza la función de un escudo. El escudo absorbe las balas y misiles, pero el láser es capaz de atravesar esta defensa y así puede conseguir dañar al jefe. El escudo, a diferencia del correspondiente de la puerta, tiene una característica de vida que, al recibir una determinada cantidad de daño, desaparecerá durante un lapso. Cuando esta situación se produzca, la rotura del escudo arrojará al jefe hacia atrás, ofreciendo, de esta manera, un tiempo extra al jugador. Cuando el jefe pierda un tercio de la vida máxima, cambiará de fase.
- Nada más comenzar la segunda pase, el escorpión jefe se proyectará hacia detrás y desde esa posición comenzará la invocación a Scorpions (finalmente, sí que se realizó el enemigo invocador). El jefe no realizará ningún movimiento, pero creará muchos escorpiones. En relación con el escudo, esta defensa seguirá el mismo comportamiento que en la primera fase a diferencia de que el escorpión no podrá retroceder más y, por ello, cuando se rompa el escudo perderá la capacidad de invocar más escorpiones durante un tiempo. Cuando el jugador consiga eliminar otro tercio más de su vida, el jefe pasará a la última fase.

- La fase final es muy similar a la primera fase, pero el Scorpion será capaz de activar su escudo de manera mucho más rápida. No solo contará con esta característica, sino que ahora dispondrá de cuatro tipos de escudo seleccionados aleatoriamente:
    - Un escudo azul, que es inmune a cohetes y balas, de cuatro segundos de duración. Sin embargo, el láser todavía será capaz de atravesarlo.
    - Un escudo rojo, que deshabilita el daño por láser, pero que recibe daño de balas y cohetes.
    - Un escudo gris, inmune a las balas, pero no al cohete.
    - Un escudo amarrillo, inmune a los cohetes, pero no a las balas.

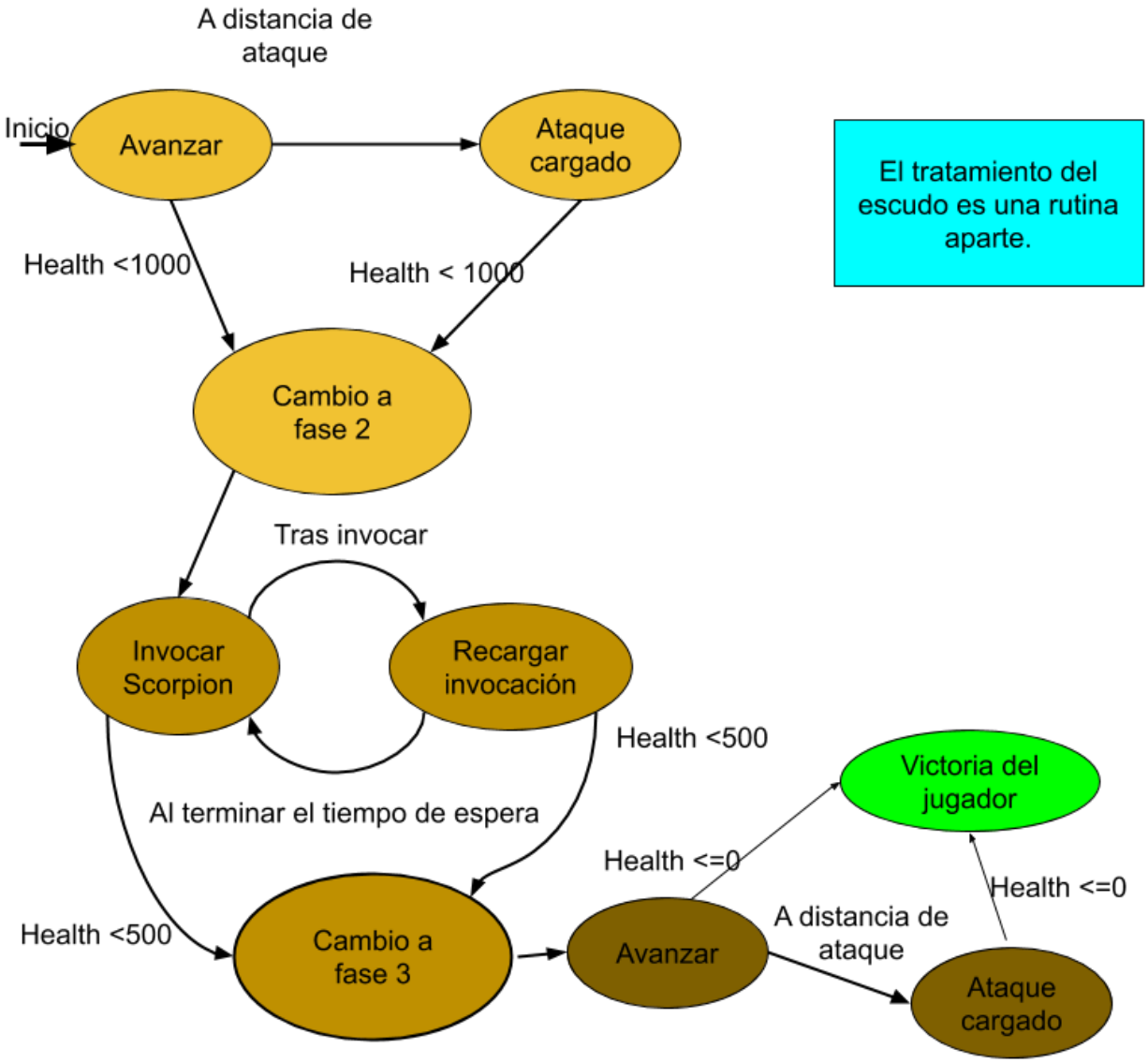


Figura 23: Lógica del jefe final.

Finalmente, cuando el jugador derrote al Scorpion jefe, saltará una escena de “Juego terminado” [Figura 24] y dará la posibilidad de seleccionar un nivel anterior o comenzar una nueva partida.

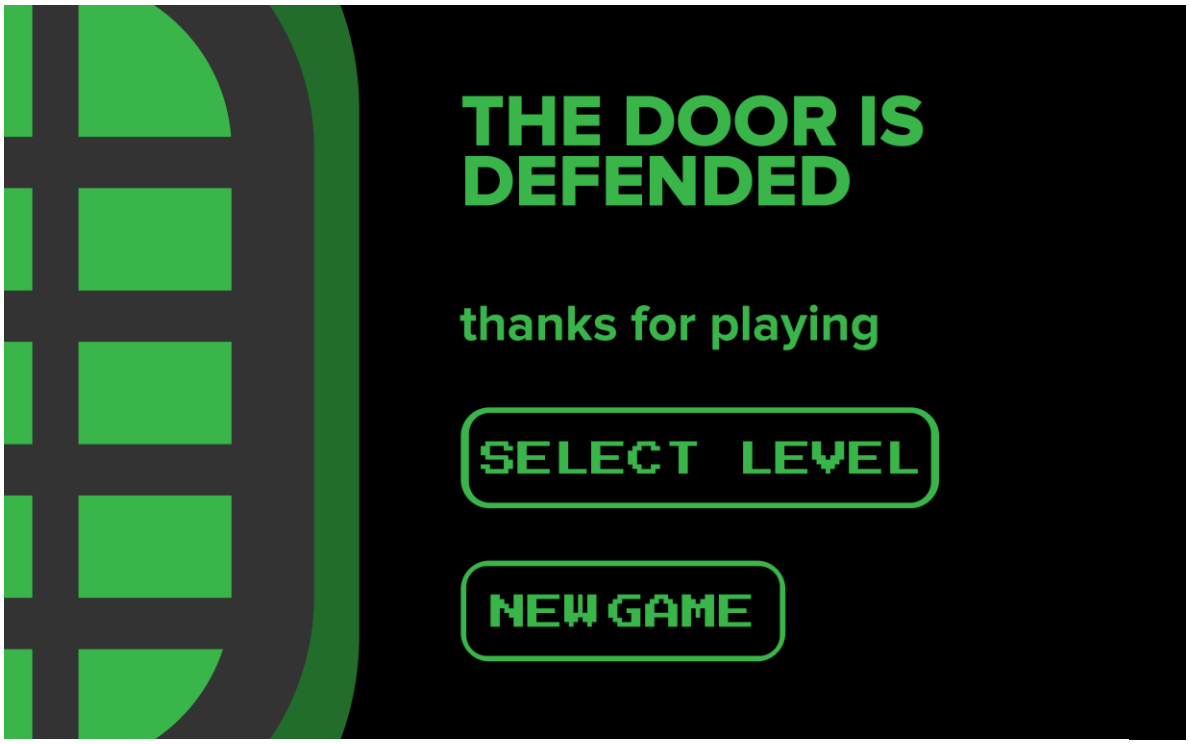


Figura 24: Pantalla mostrada tras derrotar al jefe final.

6. Pruebas

Los enemigos no han mostrado un comportamiento inesperado en ninguna de las ejecuciones realizadas. Se añadió una gran cantidad de enemigos en pantalla y el juego siguió funcionando correctamente.

La pistola dispara según las pretensiones de su diseño, pero se comprobó que en ocasiones las balas atravesaban a los enemigos, sobre todo a las Wasp. Pero este error ha ocurrido con muy poca frecuencia, y dado que sucede sobre todo con las avispas, da la sensación de que estos enemigos esquivan las balas.

El láser funciona perfectamente, no se observó ningún error en las pruebas que se realizaron.

El lanzacohetes no ha presentado problemas al lanzar los misiles, pero a veces los enemigos reciben dos veces el daño de explosión. Esto ha sucedido con muy baja frecuencia y, al ser una ventaja cuando se está jugando más que una desventaja, no representa un fallo relevante.

Las mejoras han aplicado el daño esperado y solo se permite cuando los materiales son los necesarios y establecidos en todas las pruebas realizadas.

Los niveles han funcionado correctamente, invocando a las criaturas esperadas y terminando solo cuando todos los enemigos habían sido vencidos. Todos los niveles estaban balanceados de manera que con la mejora correspondiente era posible superarlos.

El juego fue distribuido a compañeros y amigos, y funcionó correctamente en todos los equipos. La dificultad del juego fue un poco elevada, pero eso permitió el objetivo de que se probarán todas las armas y que, al final, tras entender las mecánicas y realizar las mejoras de las armas, los jugadores se sintieran poderosos al derrotar a los enemigos con extremada facilidad.

7. Resultados

Aquí se muestran algunos de los momentos más característicos del juego.

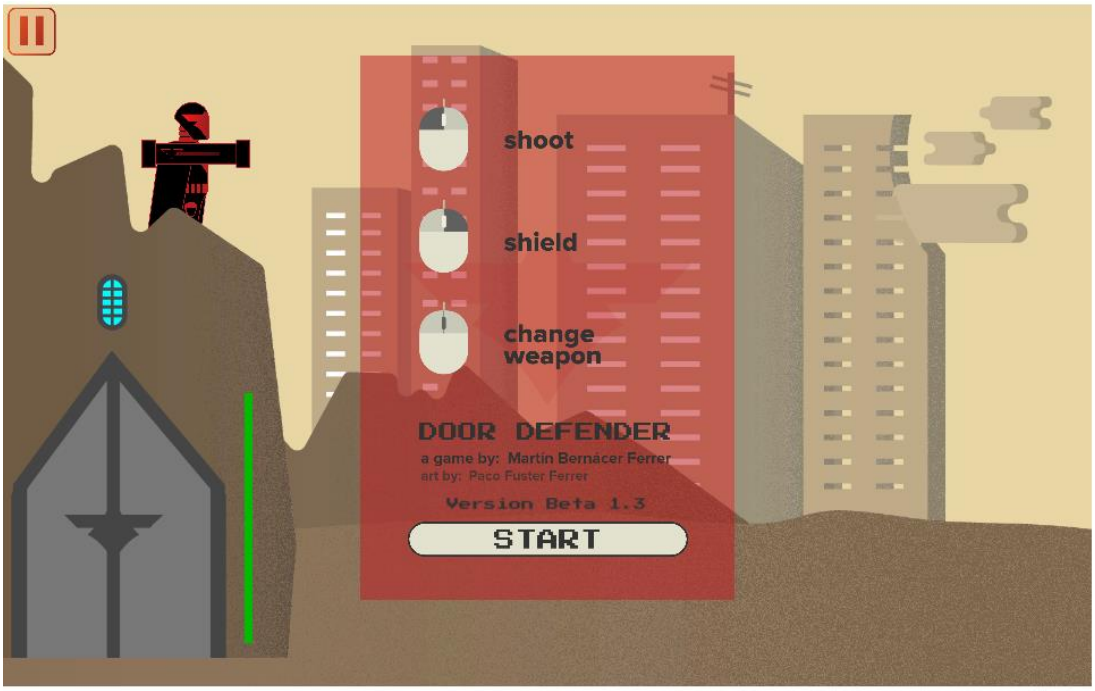


Figura 25: Pantalla de inicio del juego

[Figura 25] Muestra la pantalla justo al inicio del juego, donde se le indica al jugador los controles y las personas encargadas de realizar el juego.

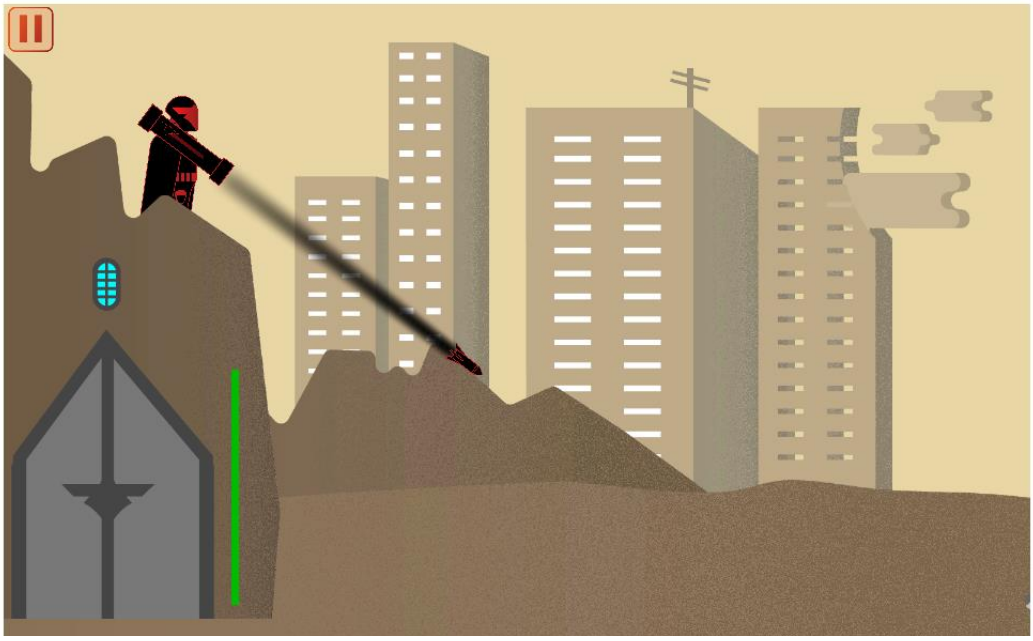


Figura 26: Cohete en movimiento.

[Figura 26] El cohete deja un haz tras de sí. Es un arma lenta, pero con mucho daño y en área.

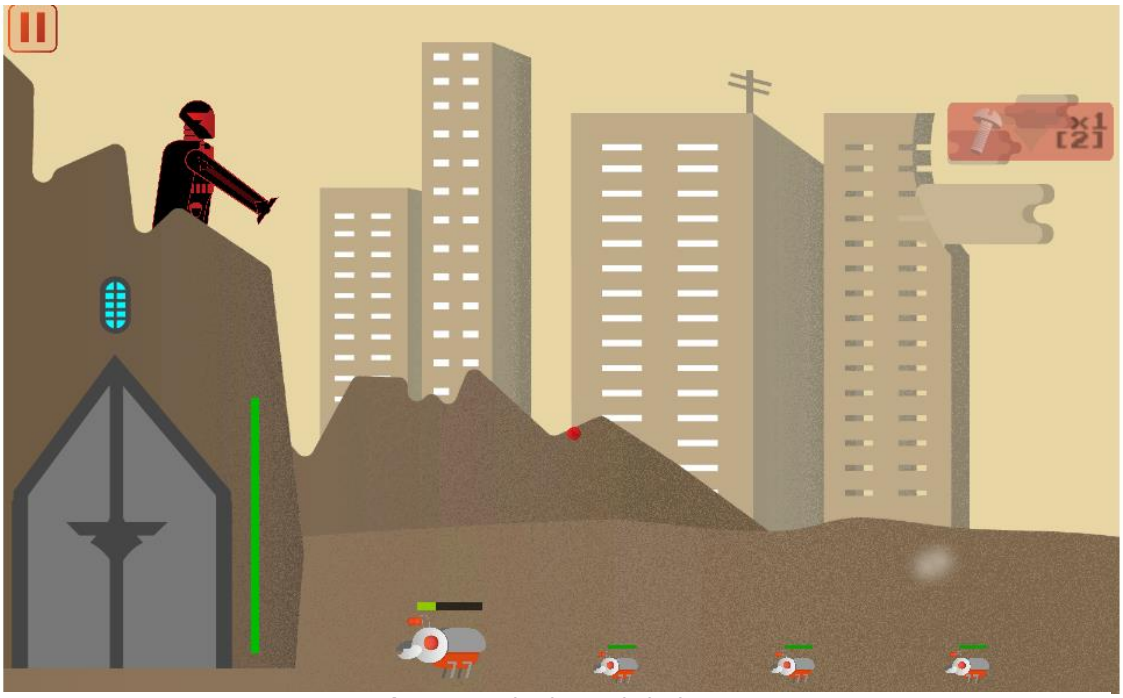


Figura 27: Pistola en el nivel 0.

[Figura 27] En el primer nivel, solo salen Cockroach, pero de diferentes tamaños. En esta imagen también podemos observar que el jugador acaba de conseguir un “Tornillo”.

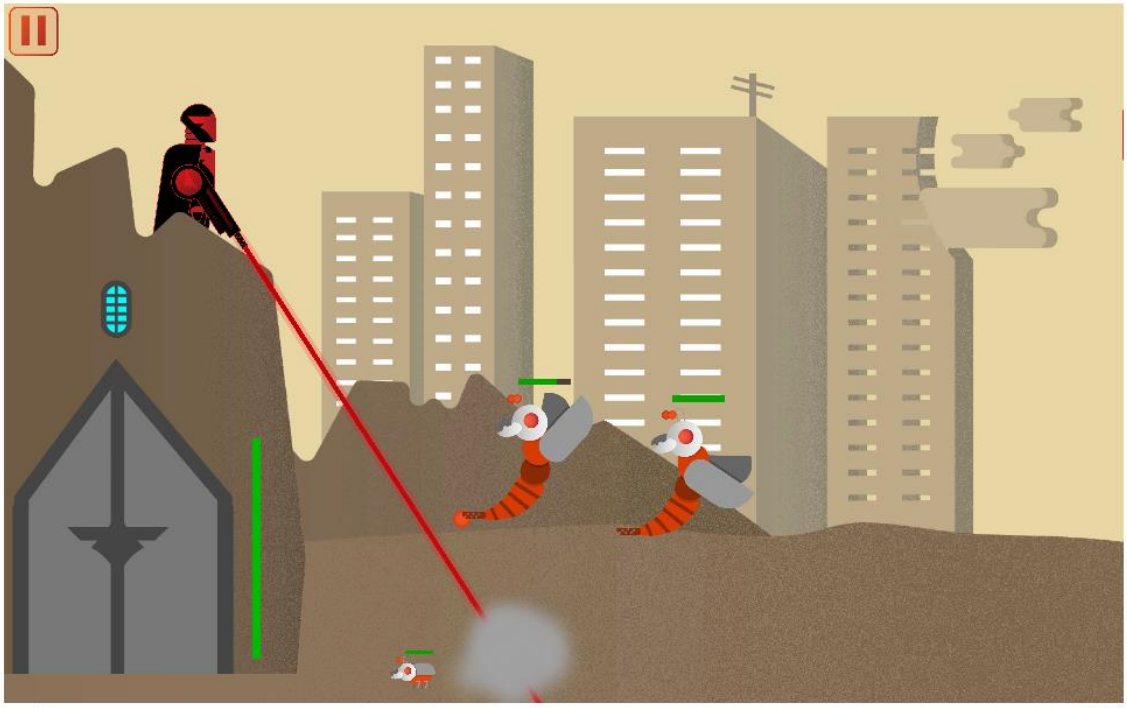


Figura 28: Arma laser en el nivel 1.

[Figura 28] El arma laser es muy útil para dañar a enemigos aéreos y terrestres a la vez. Las Wasp empiezan a aparecer a partir del nivel 1.

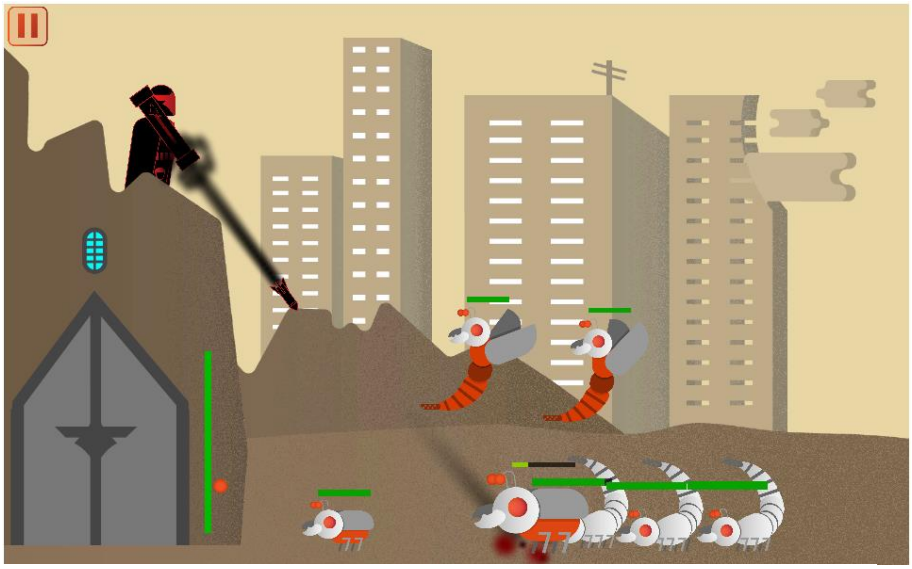


Figura 29: Cohete en nivel 2.

[Figura 29] Los Scorpion empiezan a salir en el nivel 2. Como atacan tantos enemigos, en esta situación es mejor utilizar el lanzacohetes o el láser que la pistola.

Figura 30: El jefe final en su última fase atacando la puerta con la ayuda de una de sus invocaciones.

[Figura 30] El jefe final es un enemigo formidable y no se debe tomar a la ligera. En esta captura, el jugador es probable que pierda la partida, dado que su escudo está en recarga y el jefe tiene asistencia de un aliado. La mejor estrategia es derrotar a todos los enemigos antes de pasar a la fase tres del jefe, ya que en la fase 2 el jefe es muy pasivo.

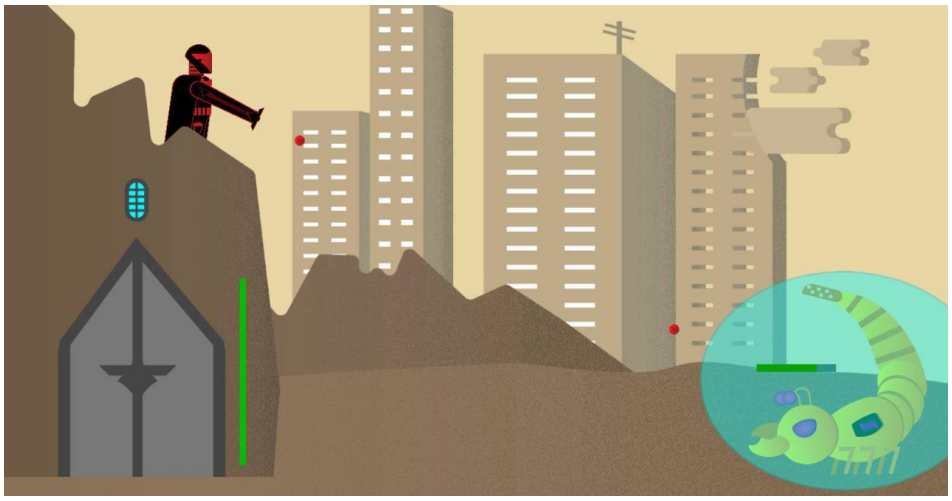


Figura 31: El jefe final utilizando su escudo en la primera fase al final del nivel 3.

[Figura 31] La primera fase es relativamente sencilla, utilizar el láser para dañarle o romper el escudo para retrasar su avance y poder perjudicarle con la pistola son dos estrategias eficaces.

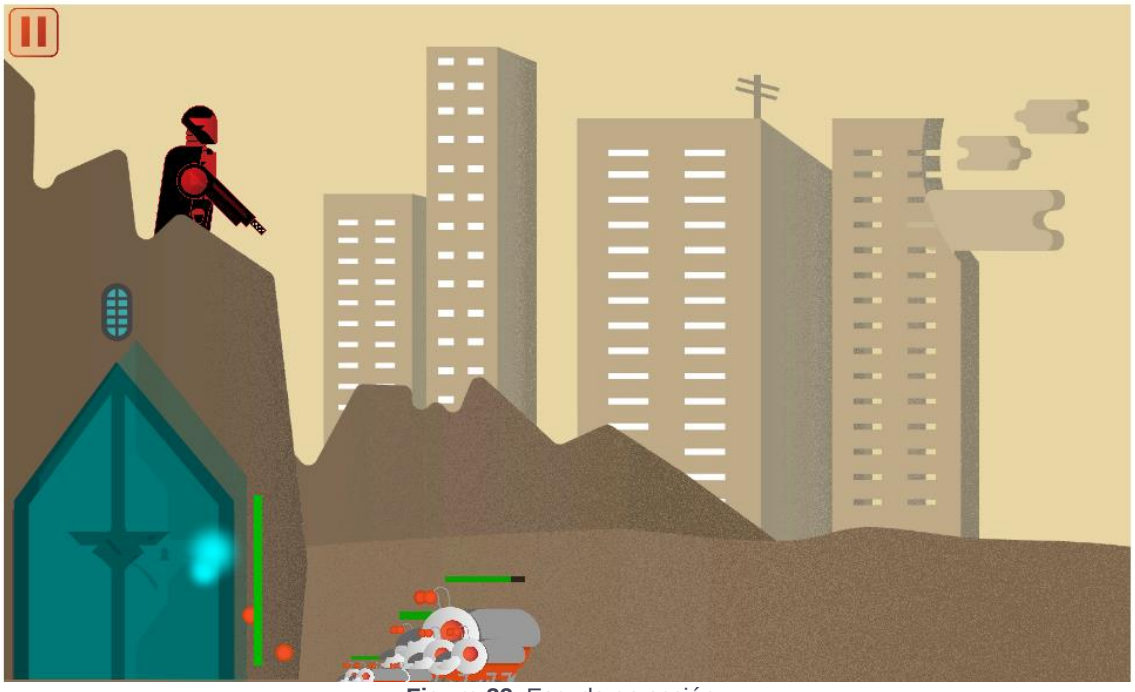


Figura 32: Escudo en acción.

[Figura 32] El escudo para cualquier proyectil. Es una defensa muy poderosa, pero su tiempo de reutilización es demasiado elevado para utilizarla de forma indiscriminada. Si te enfrentas a un enemigo con un ataque cargado, lo mejor es esperarse a que lo utilice para anular todo su ataque.

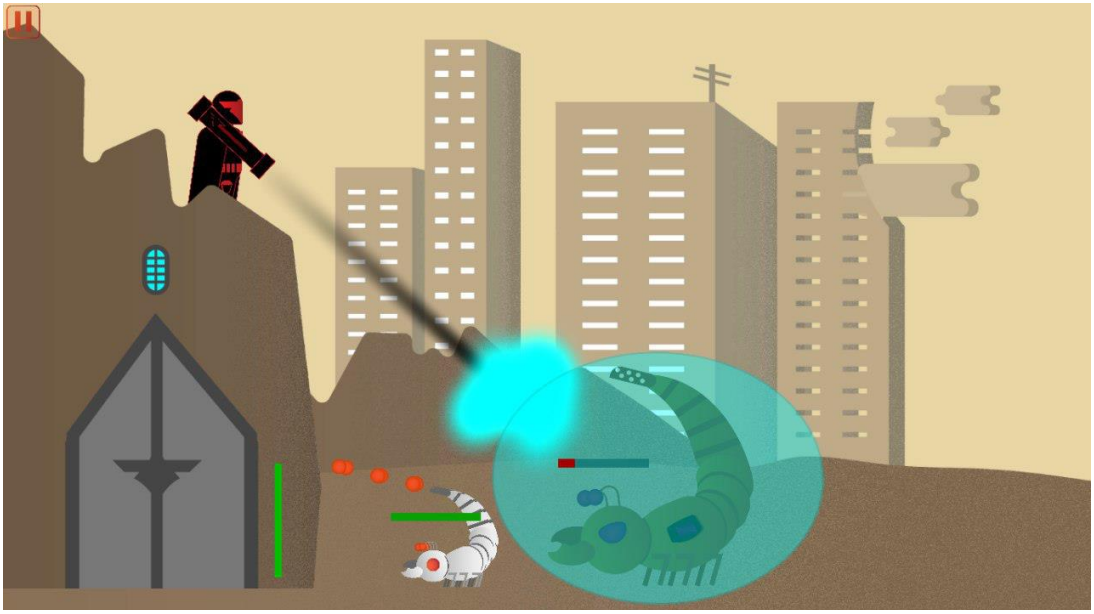


Figura 33: Escudo del jefe de la tercera fase. Inmune al cohete que el jugador está disparando.

[Figura 33] Aunque el escudo tenga un aspecto similar al de la primera fase, su comportamiento es totalmente distinto. El uso del láser es la única estrategia viable, ya que cualquier otro proyectil será eliminado sin causar ningún daño tanto al escudo como al jefe final. El escudo durará cuatro segundos y después cambiará a otro de los tres escudos.

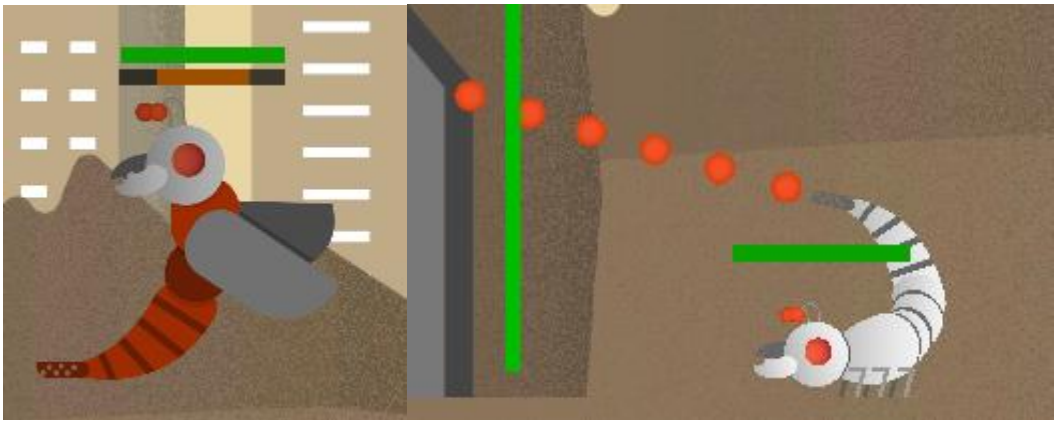


Figura 35: Fase de carga del ataque cargado y ejecución.

[Figura 35] Los ataques cargados son muy predecibles, pero muy poderosos en consecuencia. Reserva el escudo para parar o no dejes llegar a la puerta al del enemigo. El ataque cargado de Scorpion es mucho más veloz que el de los otros enemigos.

Figura 34: Todos los enemigos del juego y la comparación de sus tamaños.

[Figura 34] Normalmente, un mayor tamaño indica una mayor salud y ataque, pero menor velocidad. Cuando mayor es un enemigo, resulta más fácil de acertar. Aun así, el Scorpion es más pequeño que la Big Cockroach pero tiene diez veces más de salud.

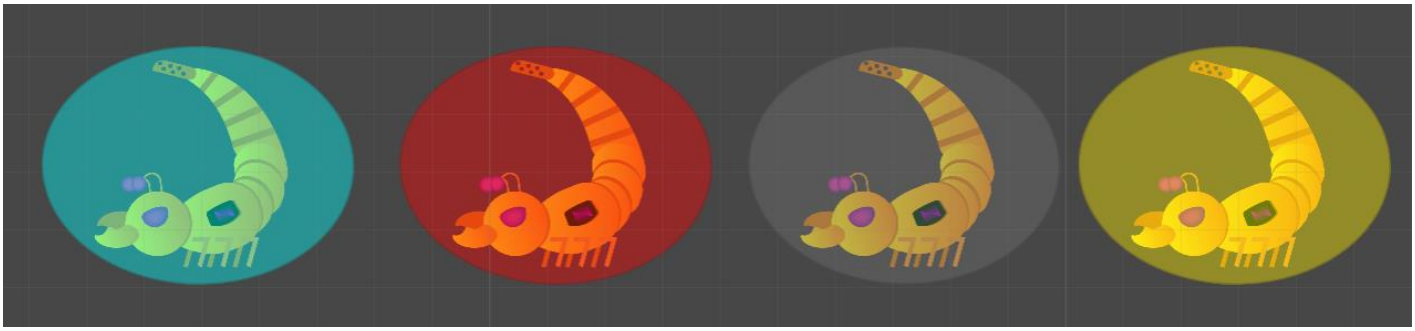


Figura 36: Los cuatro tipos de escudo de la fase 3 del jefe final.

[Figura 36] Los escudos del jefe final dan un componente de puzle a la última fase. El jugador no sabrá la función de cada escudo, pero gracias a los efectos de partículas y a la animación de daño que también dispone esta defensa, el jugador conocerá si su ataque es eficaz o fútil.

8. Conclusiones

Desarrollar un videojuego pone a prueba todos los campos de la informática. Los videojuegos implementan inteligencia artificial, gestión de información, estructuras de datos e incluso redes si son del modo multijugador.

Unity3D ha facilitado este trabajo al ser una herramienta intuitiva y ha simplificado el trabajo realizado y acortado enormemente el tiempo de desarrollo del videojuego.

La creación del videojuego ha permitido mejorar otra serie de habilidades adquiridas en el grado, como la programación orientada a objetos y la creación de algoritmos eficientes. Implementar un videojuego después de este proyecto resultará mucho más sencillo y directo y, una vez familiarizado con el entorno de Unity, las posibilidades serán mucho mayores.

El resultado final ha sido muy satisfactorio. Se han completado todos los objetivos que se habían propuesto. Tras experimentar el Door Defender se ha podido comprobar que cumple las premisas y expectativa previas, al evidenciar que dispone de:

- Una jugabilidad elevada.
- Unos enemigos consistentes.
- Efectos especiales adecuados.
- Una opción defensiva como elemento estratégico.
- Un significativo empoderamiento del jugador.
- Un clímax final que fortalece la sensación de victoria.

Hay que añadir que incluso resultó entretenido después de haber jugado con el Door Defender bajo el “modo comprobación” para determinar que el funcionamiento era el correcto. La utilización de las armas resulta una sensación única y agradable, las mejoras de las mismas son relevantes y la curva de dificultad está correctamente trazada. Los gráficos del juego también han ayudado mucho a dar un aspecto profesional.

Como curiosidad, a todos los compañeros que se les ha suministrado una copia del juego lo han disfrutado y les ha sorprendido su complejidad.

9. Trabajos futuros

Los elementos más importantes que se podrían implementar para mejorar el proyecto son los siguientes:

- Música y sonido. La intención era que los sonidos y música utilizados fueran originales. Se grabaron algunos sonidos y algunos temas, pero necesitaban ser pulidos y no se disponía de tiempo efectivo y añadirlos sin la elaboración suficiente restaría a la experiencia. Es un aspecto muy importante y aunque es cierto que muchos de los juegos en plataformas como minijuegos.com tratan el sonido en segundo plano, los jugadores que han podido probar el juego lo han echado en falta.
- Más niveles, más fondos y nuevos enemigos. El juego es corto, pero tiene recorrido para experimentar más con las mecánicas desarrolladas, quizá incluso con la adición de alguna un arma.

Además de mejorar el juego, se podría publicar en diferentes páginas. El control es lo suficiente sencillo, incluso para llevarlo al terreno de la telefonía móvil.

Otra posibilidad sería utilizar todo lo aprendido para realizar un juego de mayor escala.

10. Bibliografía

[1] AEVI. El anuario del videojuego (2018). [Internet]
Enlace: http://www.aevi.org.es/documentacion/el-anuario-del-videojuego/

[2] Molecular Psychiatry. Playing Super Mario induces structural brain plasticity: grey matter changes resulting from training with a commercial video game (2013). [Internet]
Enlace: https://www.nature.com/articles/mp2013120

[3] AEVI. El videojuego en el mundo (2017) [Internet]
Enlace: http://www.aevi.org.es/la-industria-del-videojuego/en-el-mundo/

[4] LOL: entradas de la final del split de verano ya disponibles (2019) [Internet]
Enlace: https://eu.lolesports.com/es/entradas/Conseguid-entradas-de-la-final-de-la-LEC-en-Atenas

[5] E3 2019: La feria de videojuegos más importante del mundo abre sus puertas mirando al futuro (2019) [Internet]
Enlace: https://www.20minutos.es/videojuegos/noticia/e3-feria-videojuegos-abre-mirando-futuro-3667786/0/#xtor=AD-15&xts=467263

[6] Digital. Si hay una industria que no es un juego, esa es la de los videojuegos(2018)[Internet]
Enlace: https://en.digital/blog/videojuegos-industria-mobile-crecimiento

[7] Congreso. Titulo X. De las proposiciones no de ley(2012). [Internet]
Enlace:
http://www.congreso.es/portal/page/portal/Congreso/Congreso/Hist_Normas/Norm/Reglam/T10

[8] El primer ministro de Japón se convierte en Super Mario para presentar Tokio 2020 (2016) [Internet]
Enlace:https://verne.elpais.com/verne/2016/08/22/articulo/1471846331_102160.html

[9] El pasatiempo mental salta al videojuego (2006) [Internet]
Enlace: https://elpais.com/diario/2006/03/23/ciberpais/1143084267_850215.html

[10] Metacritic. Plants vs Zombies [Internet]
Enlace: https://www.metacritic.com/game/pc/plants-vs-zombies

[11] Game Hunters. Plants vs Zombies makes iPhone killing(210) [Internet]
Enlace: http://content.usatoday.com/communities/gamehunters/post/2010/02/plants-vs-zombies-makes-an-iphone-killing/1

[12] AIAS. 13th Annual Interactive Achievement Award Finalist (2010) [PDF]

Enlace: https://web.archive.org/web/20100215035543/http://www.interactive.org/images/pdfs/13th_Annual_IAA_Finalists.pdf

[13] Metacrirtic. Pixel Junk Monsters. [Internet]
Enlace: https://www.metacritic.com/game/playstation-3/pixeljunk-monsters

[14] ArmorGames. Most Played [Internet]
Enlace: https://armorgames.com/category.php?main=all&sub=plays#games

[15] Coffee Stain Studios. Sanctum. [Internet]
Enlace: https://www.coffeestainstudios.com/games/sanctum/

[16] ZTGD. Sanctum Review (2011). [Internet]
Enlace: http://www.ztgd.com/reviews/sanctum/

[17] Game Watcher. New content coming to Dungeon Defenders 2 and Terraria thanks to a Cross-Over event (2016). [Internet]
Enlace: https://www.gamewatcher.com/news/2016-03-11-new-content-coming-to-dungeon-defenders-2-and-terraria-thanks-to-a-cross-over-event

[18] Qué es la Ludología (2015) [Internet]
Enlace: https://ludologia-bcn.blogspot.com/2015/05/que-es-la-ludologia.html

[19] OMS. La adicción a los videojuegos ya se considera un trastorno mental(2019). [Internet]
Enlace: https://www.xataka.com/videojuegos/adiccion-videojuegos-acaba-ser-reconocida-como-enfermedad-organizacion-mundial-salud

[20] Santillanaplus. La dimensión socieducativa de los videojuegos (2000) [PDF]
Enlace: https://santillanaplus.com.co/pdf/gros.pdf

[21] Saludadiario. Los beneficios del juego para la salud (2018)[Internet]
Enlace: https://www.saludadiario.es/vademecum/los-beneficios-del-juego-para-la-salud

[22] Meristation. Los videojuegos ayudan a mejorar a los cirujanos (2012) [Internet]
Enlace: https://as.com/meristation/2007/02/20/noticias/1171965720_065645.html

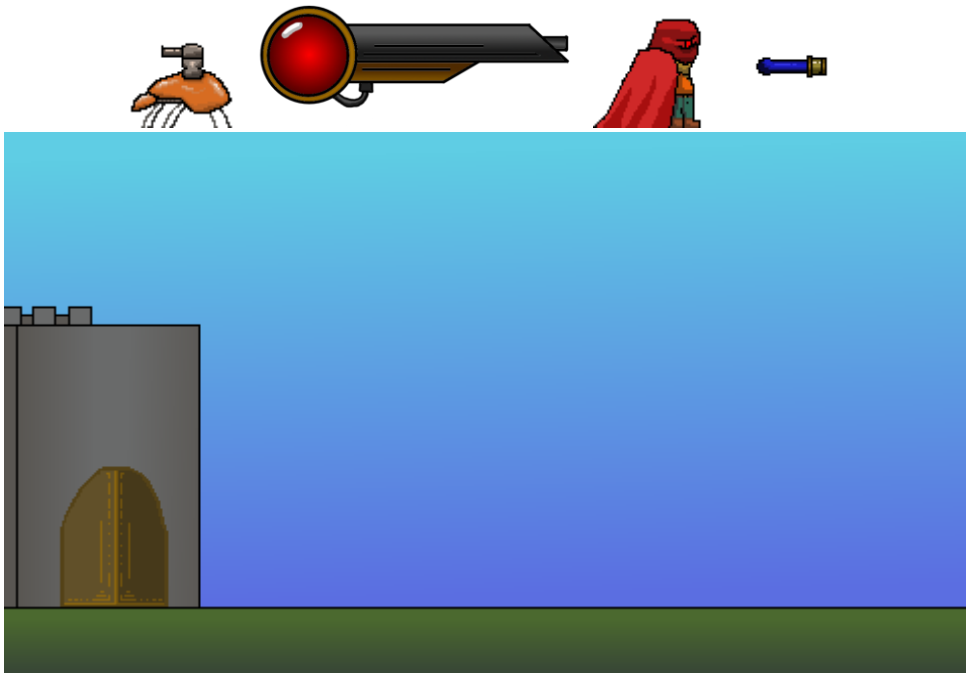
[23] Unity 3D. [Internet]
Enlace: https://unity3d.com/es/unity

[24] Unity Documentation. User Manual. [Internet]
Enlace: https://docs.unity3d.com/Manual/index.html

[25] Unity Documentation. Unity Scripting Reference. [Internet]
Enlace: https://docs.unity3d.com/ScriptReference/index.html

11. Anexo: Recursos del juego

11.1 Bocetos originales



Boceto Fondo

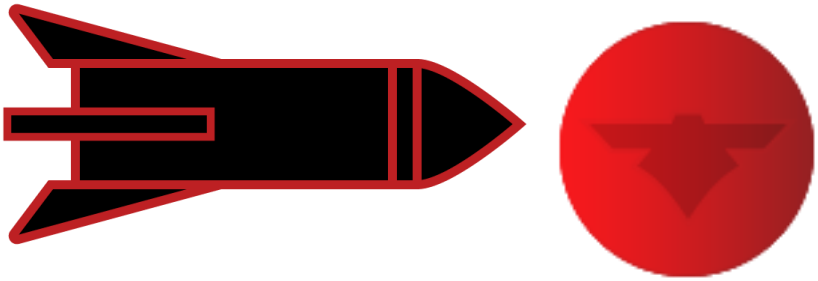
11.2 Armas

Arma Laser

Lanza cohetes

Pistola

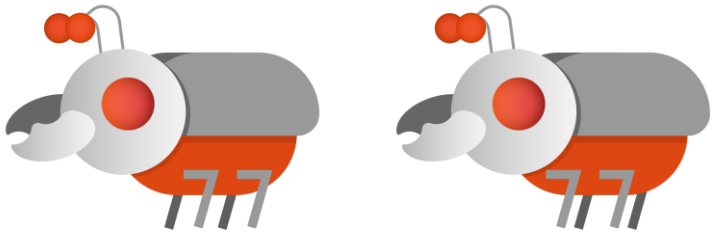
Laser



Cohete

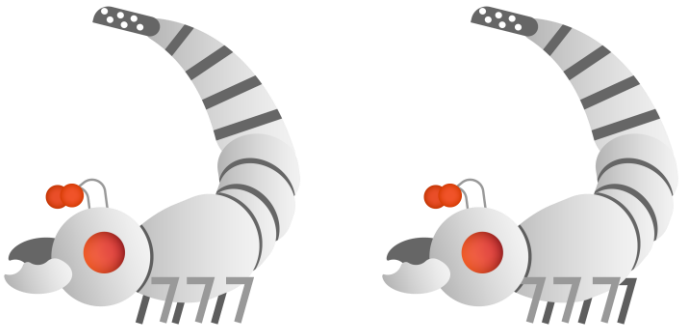
Bala

11.3 Enemigos



Cockroach

Wasp

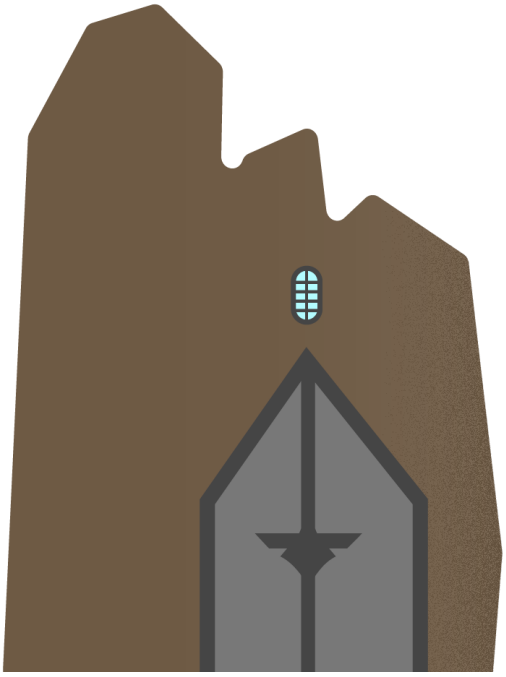


Scorpion

Scorpion Boss

11.4 Escena

Escenario

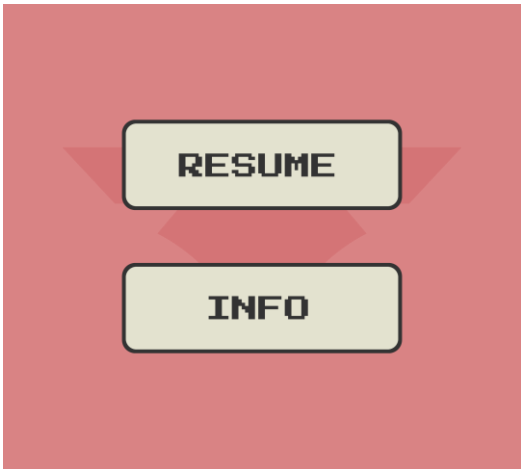


Puerta

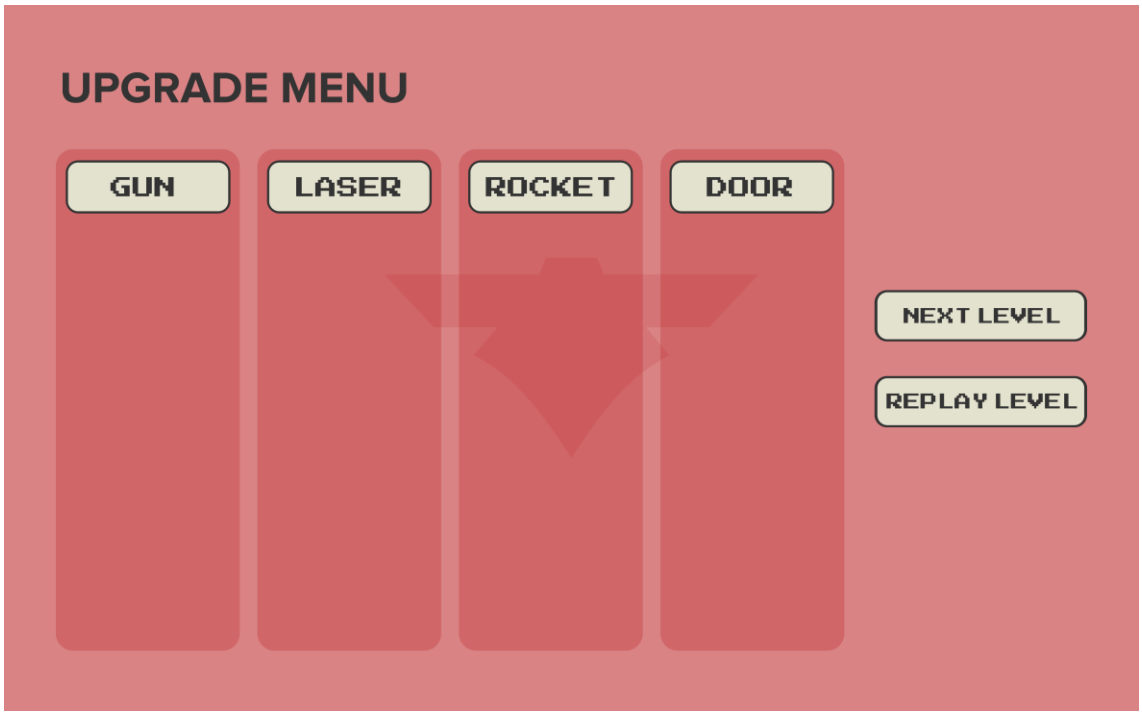
Escudo

11.5 Interfaces

Menú de información

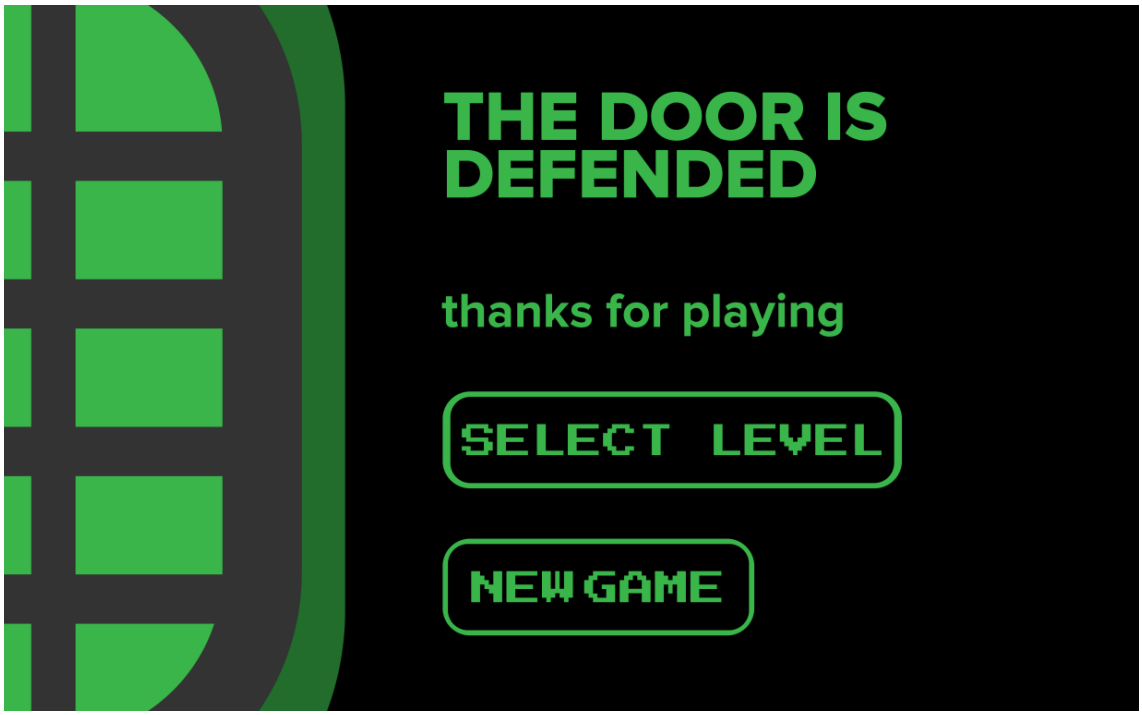


Menú de opciones/ Menú de pausa



Menú de mejoras

Fin del juego

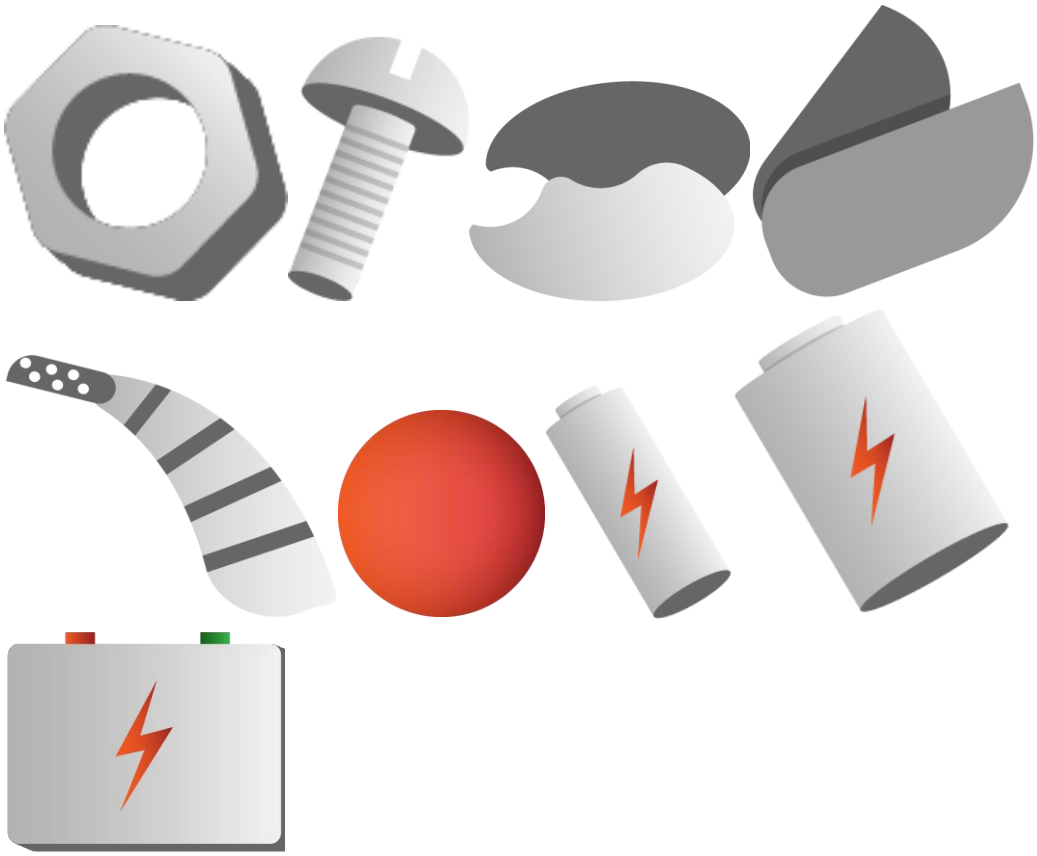


Juego terminado

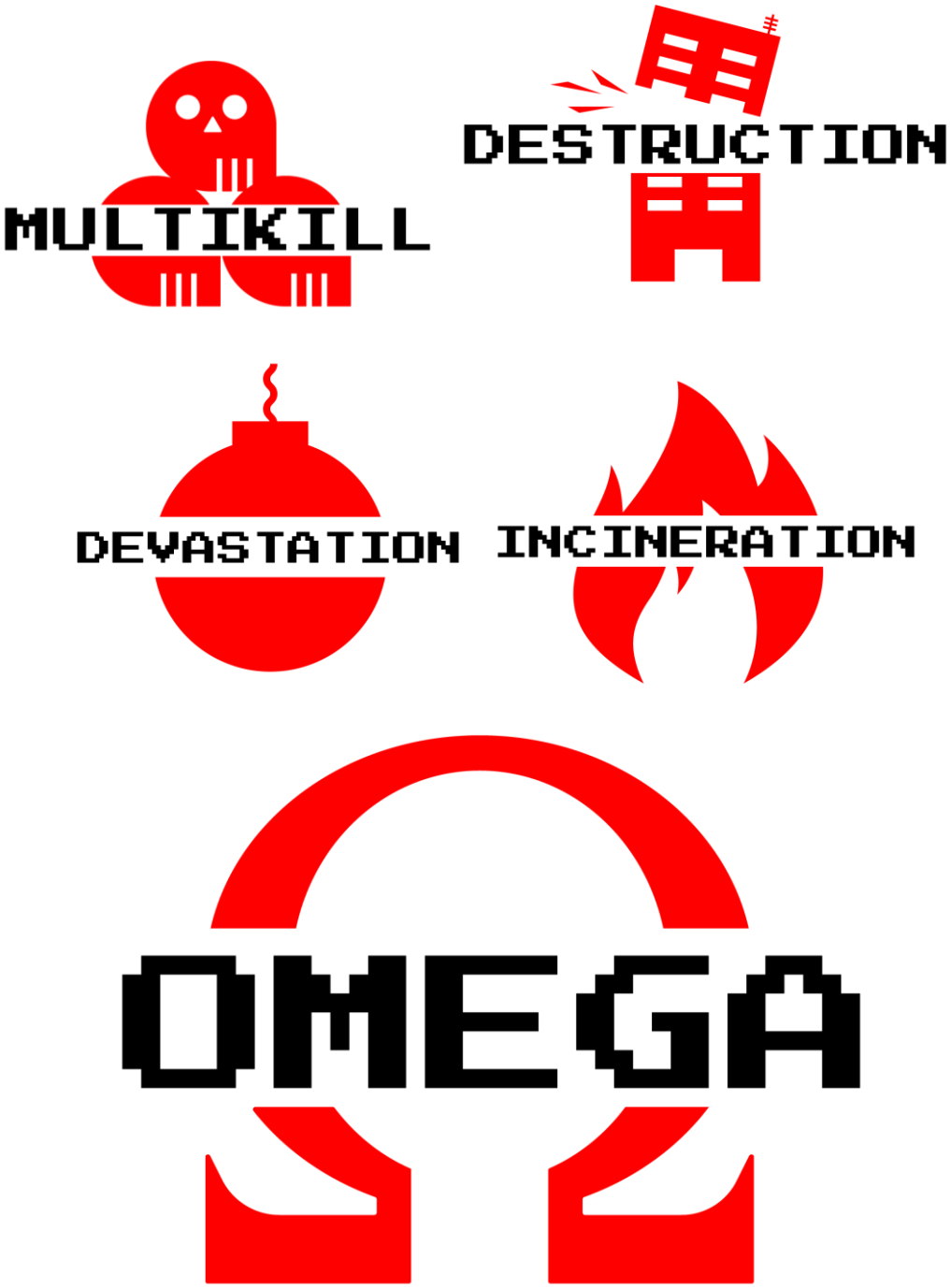
Nuevo objeto

Botón de opciones (en desuso) /Botón de pausa

11.6 *Items*



11.7 Killstreak



12. Glosario

- Array: Lista de elementos
- Curva de dificultad: En un videojuego, la dificultad va subiendo a medida que el jugador avanza.
- Float: Tipo de variable de número con decimales.
- Int: Tipo de variable de número entero.
- Shooter: Genero de videojuego centrado en disparar. Puede ser en primera persona, en tercera o en 2D.
- String: Tipo de variable de secuencias de caracteres.
- Tower defense: Género de videojuegos de estrategia. Consiste normalmente en resistir oleadas a base de construir torretas que ataquen automáticamente a los enemigos. Las oleadas son hordas de enemigos, que intentan destruir un nexo u objeto valioso, ignorando las torretas y centrándose a destruir el objetivo que desea proteger el jugador.