Creación de un videojuego de género shooter para móvil y sobremesa

Universidad de las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Pablo José Díaz García

Tutor: Agustín Trujillo Pino

Las Palmas de Gran Canaria, Julio de 2017

Contenido

1.- Motivaciones y objetivos del proyecto

En este trabajo de fin de grado, se pretende realizar un prototipo jugable de un videojuego para ordenador de sobremesa y para dispositivos móviles Android. Todo el desarrollo seguido, así como las dificultades que han ido apareciendo se han recopilado en esta memoria.

Para alcanzar esta meta, se han tenido que ir cumpliendo una serie de objetivos durante el desarrollo:

- Identificación de los diferentes motores de videojuegos que existen actualmente, para ver cual es más acorde con nuestro proyecto.
- Identificación y aprendizaje de las diferentes técnicas de diseño de videojuegos para así poder aplicar estas prácticas a nuestro juego.
- Plantear unas mecánicas base pero robustas antes de la creación de nuestro juego.
- Crear prototipos para cada mecánica de nuestro juego e ir probándolos continuamente.
- Análisis y pruebas de las diferentes plataformas de control que usara nuestro juego (ratón, teclado y pantalla táctil).

La elección de un videojuego como trabajo de fin de grado ha sido en gran parte por la atracción que tengo por ellos desde hace bastantes años y por las facilidades que existen hoy en día para publicar tu trabajo y poder introducirte poco a poco en el mercado (google play, greenlight de steam etc.).

2.- Justificación de las competencias

En este apartado hablaremos de las diferentes competencias que se han cubierto durante la realización de este trabajo de fin de grado, así como una pequeña explicación de las mismas.

2.1.- Competencias comunes a la ingeniería informática

CII01 - Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

Esta competencia ha sido cubierta durante todas las fases del proyecto (análisis y desarrollo), ya que continuamente se ha ido comprobando la calidad y seguridad de los elementos generados mediante las pruebas.

CII02 - Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

Como podemos observar en el apartado “desarrollo” y en “pruebas y corrección” de esta memoria, todos estos objetivos de esta competencia se han ido cumpliendo.

CII010 - Conocimiento de las características, funcionalidades y estructura de los Sistemas Operativos y diseñar e Implementar aplicaciones basadas en sus servicios.

Esta competencia ha sido cubierta, ya que al realizar el juego tanto para Windows como para Android, hemos debido conocer las funcionalidades y características de ambos SO, ya que se han tenido que realizar cambios en el juegos según el SO al que fuese destinado.

CII015 - Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica.

Esta competencia ha sido cubierta de forma básica a la hora de diseñar el comportamiento y los enemigos y el ataque de multitudes.

2.2.- Competencias de la intensificación de ingeniería del software

IS01 - Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.

Como podemos observar, todos los requisitos que habían sido expuestos mediante el TFT01 han sido realizados. Además, se puede asegurar su calidad ya que han sido probados continuamente como podemos observar en el apartado de “pruebas y correcciones”.

IS02 - Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones.

Como podemos observar en el apartado "desarrollo", han ido surgiendo diferentes problemas durante el desarrollo, pero se han solucionado intentando perder el menor tiempo posible.

IS04 - Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.

Como podemos ver a lo largo de esta memoria, se han identificado y analizado varios problemas que han ido apareciendo y tanto su solución como verificación han sido documentados en esta memoria.

3.- Aportaciones al entorno social

La industria del videojuego en España es un sector que está en continuo crecimiento. Esto lo podemos observar en los 510 millones de euros que facturo a lo largo de 2015, según los datos presentados en el Libro Blando del Desarrollo Español de Videojuegos 2016, promovido por la Asociación española de empresas productoras y desarrolladoras de videojuegos y software de entretenimiento. Un 24% más que durante el año 2015. Además, se estima un crecimiento anual de 22%.

Gracias a la industria del videojuego, en España se ha incrementado un 32% el empleo para este sector. Además, se estima un crecimiento anual del 22.7%.

Respecto al panorama internacional, España es uno de los países con mayor número de empresas en este sector. Sin embargo, no es uno de los países que más ingresos genere gracias a ello. Esto se debe en su mayoría, a que es un sector que aun es "reciente" en nuestro país.

PAIS	EMPRESAS	FACTURACIÓN (M €)
EEUU	SIN DATO	3.700
FRANCIA	250	3.677
CANADA	472	2.800
ALEMANIA	320	1.820
REINO UNIDO	1092	1.490
SUECIA	213	952
FINLANDIA	260	800
ESPANA	480	511
PAISES BAJOS	352	190
DINAMARCA	171	148
BELGICA	SIN DATO	41

Imagen 1: "Ingresos generados por las empresas"
Datos extraídos del libro blanco

3.1.- Facturación según el modelo de negocio

Actualmente en España una gran parte de las ventas de videojuegos procede de las ventas digitales, seguidas por el modelo free to play + publicidad y del modelo free to play + compras en el propio juego. Conforme van pasando los años, las ventas físicas van disminuyendo y aumentan las citadas anteriormente.

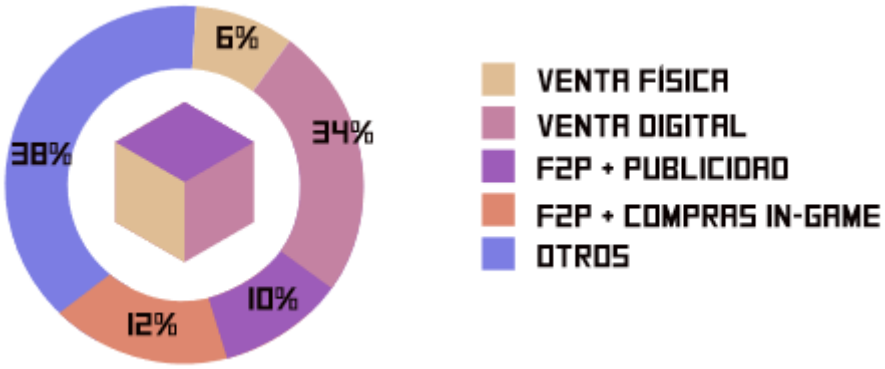


Imagen 2: "Facturaciones según el modelo de negocio"
Datos extraídos del libro blanco

3.2.- ¿Importamos o exportamos?

La industria española se dedica en gran medida a la venta internacional, el 52% de la facturación total proviene de otros países. En la actualidad es muy fácil dedicarte al mercado internacional haciendo uso de cualquiera de las plataformas existentes (steam, google play etc.) y gracias a ello la industria en España ingresa una importante cantidad de dinero.

En la actualidad, uno de los retos de la industria es aumentar el número de ventas en el ámbito nacional.

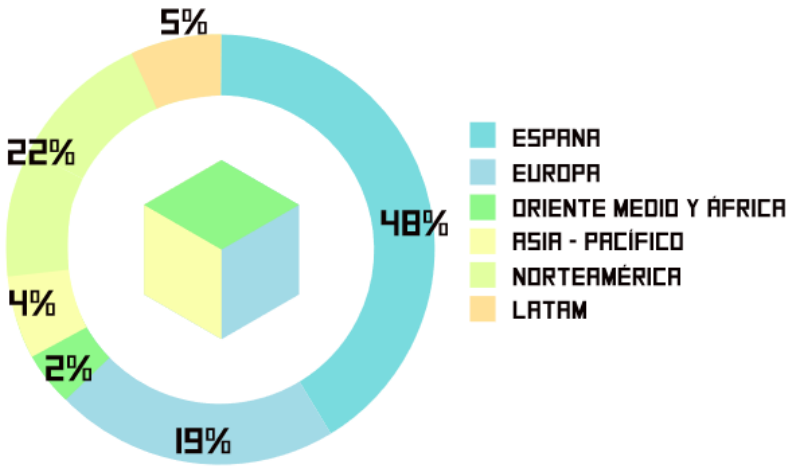


Imagen 3: "Distribución de la facturación generada por las empresas de videojuegos"
Datos extraídos del libro blanco

3.3.- Empresas españolas

En España hay censadas 480 empresas de videojuegos en activo, un 20% más que 2015. Además, existen alrededor de 125 iniciativas y proyectos empresariales, a la espera de consolidarse como empresas en el corto o medio plazo.

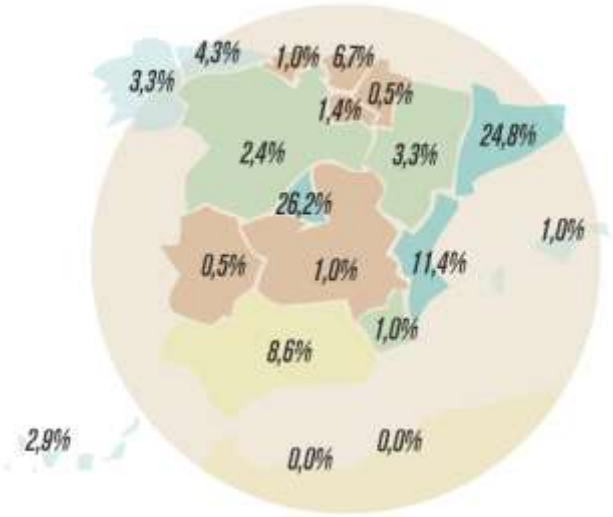


Imagen 4: "Distribución territorial de las empresas de videojuegos en España"
Datos extraídos del libro blanco

4.- La industria de los videojuegos

4.1.- Generaciones de consolas

1ª Generación

La primera generación destaca por ser capaz de llevar todos aquellos juegos que se encontraban en las maquinas recreativas a los hogares.

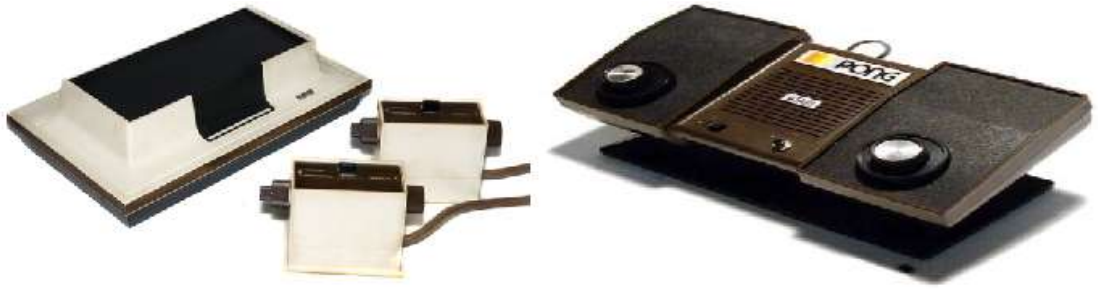


Imagen 5: "De izquierda a derecha: Magnavox Odyssey, Atari Pong"

Magnavox Odyssey

Fue la primera videoconsola en llegar a los hogares. Se empezó a comercializar en América a partir de 1972. En el primer año vendió unas 100.000 unidades a 100$.

La consola ya contaba con un sistema de cartuchos intercambiables, permitiendo así disfrutar de una gran variedad de juegos con una única consola. La mayoría de estos juegos estaban basados en los deportes.

Hay que destacar que la consola carecía de sonido y no contaba con ningún medio de almacenaje, por lo que no se podían guardar las partidas.

Atari pong

A diferencia de la Odyssey, esta consola únicamente permitía jugar a un único juego: el pong. Este juego era muy parecido al 'Table tennis' de magnavox, por lo que hubo una demanda por parte de esta última compañía por infracción de patentes. Finalmente se consiguió llegar a un acuerdo y se pudo seguir comercializando la consola. Llegando a vender unas 55.000 unidades en el año 1975.

2ª Generación

Las principales novedades durante esta generación fue la introducción de los joystick (mejorando la jugabilidad) y de la memoria ROM, permitiendo crear juegos más complejos.

Imagen 6: "De izquierda a derecha: atari 2600, sega sg 1000"

Atari 2600

Esta consola venia acompañada de dos joystick y un cartucho con juegos. A diferencia de la Atari pong, permitía jugar a diferentes juegos haciendo uso de los cartuchos (idea copiada de la magnovox Odyssey) y contaba con un sistema de sonido.

Atari recibió un duro golpe durante esta generación, ya que creó un juego basado en la película ET, pero su pésima jugabilidad y la falta de objetivos hizo que fuera un fracaso en las ventas.

Sega SG 1000

Una nueva compañía se adentraba en el mundo de los videojuegos y lo hacía desde Japón en 1981.

Esta consola usaba un sistema de cartuchos intercambiables para sus juegos. Mostraba hasta 16 colores y poseía un sistema básico de sonido.

3ª Generación

Debido al fracaso de atari durante la generación anterior, las consolas de compañías japonesas pudieron abrirse un hueco en el mercado. Durante esta generación destacaron 2 empresas: nintendo y sega.

Imagen 7: "De izquierda a derecha: NES, sega master system"

NES

Se empezó a comercializar en Japón a partir de 1983, sin embargo, no llego a Europa hasta 1986, siendo conocida bajo el nombre de “NES”.

Nintendo creó un nuevo modelo de negocio muy usado hoy en día: la concesión de licencias a terceras empresas para que desarrollasen juegos para su videoconsola. Algunas de estas licencias creadas por la empresa fueron: Zelda, Mario Bros y Donkey Kong.

Además, con esta consola se introdujo la posibilidad de guardar partidas mediante la introducción de una pequeña memoria en los cartuchos, la cual era alimentada mediante una pila de reloj.

Sega Master System

Creada por los japoneses en el 85 y lanzada en Europa durante el 89. Fue una consola que en Europa no alcanzo el mismo nivel de ventas que las NES a pesar de tener mejores características técnicas. Esto fue debido a su salida más tardía.

En este consola destacan juegos como “Sonic the hedgehog” y “Alex Kidd”.

4ª Generación

Sigue la competencia entre Sega y Nintendo por hacerse con el mercado. Además se introducen nuevas novedades como el cambio de cartuchos por CDs y el paso de los 8 a 16 bits, permitiendo una mejora en los gráficos y capacidad de un mayor almacenamiento de datos.

Imagen 8: “De izquierda a derecha: Sega mega drive, SNES”

Sega Mega Drive

Lanzada en Japón en 1988 y en Europa en 1990. Fue una consola que intento destacar por su potencia de hardware. Siendo la primera consola que usaba 16 bits reales.

En futuras actualizaciones de esta consola, se lanzaron modelos que permitían las lecturas de CDs y una consola que permitía usar 32 bits, lo cual permitió a Sega dar el salto a los juegos en 3D. Sin embargo, estas revisiones de las consolas eran muy caras y no cuajaron en el mercado.

SNES

Fue la triunfadora de la época en Japón (en Europa mandaba mega drive). Algunas de las novedades que introdujo nintendo fueron:

- Modo online mediante un modem por satélite.
- Un adaptador que permitía jugar juegos de consola portátil en la SNES. Lo que permitía reproducirlos a color (en la portátil eran en blanco y negro).

5ª Generación

Pasamos de los 18 a los 32 bits. Lo que permitió introducir los entornos en 3D, que poco a poco fueron sustituyendo al 2D.

Imagen 9: "De izquierda a derecha: PlayStation, Nintendo 64"

PlayStation

Fue la primera consola lanzada por Sony y fue todo un éxito en el mercado. Alcanzando las 100 millones de unidades vendidas, además de ser la más popular.

En esta consola se empezaron a lanzar juegos más largos y por ello Sony desarrollo unas tarjetas de memoria que permitía guardar partidas para luego continuarlas por ese mismo punto.

Nintendo 64

Lanzada en Japón y América en 1996. Fue la primera consola que usaba 64 bits y contaba con un procesador más potente que la PlayStation.

Sus cartuchos no eran de gran capacidad comparados con un CD. Sin embargo, eran más rápidos, llegando incluso a permitirles prescindir de pantallas de carga, cosa que los CD no permitían. Las partidas podían ser guardadas en el propio cartucho, no era necesario comprar memorias para guardar las partidas como ocurría con Sony.

Otra gran innovación de nintendo fue la introducción de la vibración en los mandos. Algo que más adelante fue copiado por casi todas las consolas.

6ª Generación

Durante esta generación surgen las consolas de 128 bits y aparecen nuevas características como la estandarización de los CDs como medios de distribución de juegos. Además, surgen las conexiones de red en las consolas, que les permitirían jugar en línea.

Imagen 10: "De izquierda a derecha: PS2, Xbox, GameCube"

PlayStation 2

Empezó a venderse en el 2000 en Japón, convirtiéndose en la consola más vendida y usada de esta generación. Las ventas de esta consola fueron increíbles, alcanzando las 155 millones de unidades vendidas. En parte el éxito de esta consola fue gracias al éxito de su predecesora y a la compatibilidad con todos los juegos de esta.

Mas adelante se saco una nueva revisión conocida como "Slim", que fue la primera consola de Sony que conto con disco duro y conexiones Ethernet para jugar en línea.

Xbox

Con esta consola Microsoft entra en el mundo de las consolas en el 2001. La consola se utilizo por usar una arquitectura similar a los ordenadores de la época, lo que permitió adaptar muchos de sus títulos entre plataformas de forma sencilla.

Microsoft creo Xbox Live que era una plataforma donde sus jugadores podían jugar online y descargar contenido que podían almacenar en su disco duro.

Nintendo GameCube

Lanzada en 2001 en Japón. Nintendo decide pasarse del cartucho al CD, no permitiendo en esta nueva consola la compatibilidad de los juegos de sus anteriores consolas. Además, para evitar la piratería crear su propio CD, que permitía almacenar 1.5 Gb. Lo cual por otro lado fue un inconveniente, ya que la consola no podía ser usada como reproductora de CDs estandarizados.

Fue la consola más potente de la generación, sin embargo no por ello tenia juegos con los mejores gráficos. Unos de sus grandes atractivos en que permitía la interactuación con la consola portátil de nintendo (game boy advance).

7ª Generación

En esta generación de introducen novedades como CPUs con tecnología multinucleo, GPUs sofisticadas, el uso de discos Blu-ray, el uso del wifi y la introducción de los mandos inalámbricos.

Imagen 11: "De izquierda a derecha: Xbox 360, PS3, Wii"

Xbox 360

Lanzada en 2005. Inicialmente conto con 2 versiones, que se diferenciaban por venir o no incluido un disco duro. Más tarde, sacaron una nueva revisión que permitía la conectividad Wifi.

La plataforma Xbox Live sufre cambios y añade 2 tipos de servicios. El "silver" que era gratuito y el "gold" que era de pago y permitía el modo multijugador.

PlayStation 3

Lanzada por Sony en el 2006. A lo largo de la generación se crearon 3 modelos: original, slim y super slim. Fue una consola que mejoro la PS2 y permitió visualizar contenido multimedia y navegar por internet. Además, se actualizaba automáticamente.

Sony decide copiar a Microsoft y lanza su propia plataforma online "Playstation Network", a través de la cual una vez te registrabas podías comprar juegos, películas, jugar online etc.

Su primera versión fue no tuvo gran éxito, ya que su precio era muy elevado y apenas tenía juegos. Sin embargo, con el paso del tiempo y la introducción de nuevos títulos y revisiones en la consola, esto cambio y se convirtió en un éxito de ventas (aunque no llego a alcanzar a PS2).

Wii

Lanzada en el 2006, fue la primera consola que permitía involucrar de lleno al jugador en los juegos, ya que estos funcionaban en su mayoría por el movimiento del jugador. La mayoría de juegos de esta consola estaban pensados para el multijugador offline.

La consola contaba con una gran cantidad de periféricos que simulaban los elementos usados en sus juegos: espadas, guitarras, pistolas etc. Fue una consola que llamo bastante la atención debido a sus novedades y por su bajo precio, lo que permitió a Nintendo remontar tras el fracaso de la GameCube.

8ª Generación

En la generación que estamos viviendo actualmente y donde podemos encontrar 3 grandes competidores: Sony, Microsoft y Nintendo.

Ahora mismo se podría decir que Sony lidera sin mucha dificultad el mercado, sin embargo, Microsoft poco a poco está añadiendo características muy importantes que no tiene Sony. Como la compatibilidad de forma gratuita con sus juegos de las Xbox360 y la compatibilidad entre PC y consola.

Respecto a Nintendo, es la compañía que sigue triunfando en las consolas portátiles gracias a su gran variedad de títulos. Sin embargo, en las consolas de sobremesa se podría decir que en esta generación no ha empezado con muy buen pie. Ya que a lo largo de la generación ha sacado 2 consolas totalmente diferentes: Wii U y Switch. La primera sufrió un importante fracaso debido a la poca cantidad de títulos que saco durante su escaso tiempo de vida. Y la segunda acaba de salir, así que esperemos que Nintendo se ponga las pilas y remonte, ya que cuanta más competencia exista en este sector mejor nos viene a los clientes.

4.2.- Motores de videojuegos

Al igual que ocurre en otras áreas, a lo largo de los años han ido apareciendo nuevas herramientas que facilitan el desarrollo de una tarea. En la programación web se conocen como frameworks mientras que en el ámbito de la creación de los videojuegos se conocen como motores de videojuegos. Gracias a ellos, podemos realizar muchas tareas de forma sencilla sin necesidad de realizar tareas de bajo nivel.

Algunos de los motores más destacados que existen en la actualidad son los siguientes:

Unity 5

Imagen 12: "Logo Unity"

Uno de los motores gráficos más usado entre los que empiezan a desarrollar en el mundo de los videojuegos es Unity.

Unity 5 destaca por su facilidad para portar un proyecto a otras plataformas sin necesidad de estar creando nuevos proyectos o cambiando código, lo cual facilita enormemente la tarea de crear nuevas versiones de un juego para otras plataformas. Además, tiene soporte para Oculus Rift.

Una de las grandes ventajas de Unity respecto a otros motores, es además que tiene una versión gratuita. Para desarrollar en Unity podremos usar JavaScript, C# y Boo.

Algunos de los últimos juegos lanzados y que han sido desarrollados con esta plataforma son: Furi, superhot, super Mario run y Ori and the blind forest.

CryEngine

Imagen 13: "Logo CryEngine"

Motor de juegos creado por la compañía Crytek. Su gran fama ha sido adquirida gracias a juegos como 'Crysis' o 'Ryse: son of rome', que destacan por sus gráficos realistas y los efectos de iluminación.

Unas de las principales características por las que destaca este motor, es por la gran cantidad de herramientas que ofrece a la hora de crear escenarios de una calidad asombrosa.

La curva de aprendizaje del motor puede ser algo complicada, por lo tanto la carga de trabajo para grupos de 1 o 2 personas puede llegar a ser frustrante.

Unreal Engine

Imagen 14: "Logo unreal engine"

Desarrollado por Epic Games. Este motor vio su primera versión en 1998. En la actualidad es uno de los motores más usados para juegos AAA, tanto para PC como para consolas. Aunque también es usado para juegos Indies, pero no tanto como Unity.

La curva de aprendizaje no es tan alta como Cryengine, pero no llega a ser tan fácil como la de Unity y el lenguaje que usa es C++.

Algunos de los juegos desarrollados con este motor son: 'Mortal Kombat X', 'Dead Island 2' y 'Life Is Strange'.

Construct 2

Imagen 15: "Logo construct 2"

Desarrollado por Scirra y centrado principalmente en el entorno HTML5. Esto es debido, a que existen numerosas maneras para convertir HTML5 en aplicaciones para las diferentes plataformas (Windows, Android, navegadores etc.).

Construct 2 se caracteriza por permitir crear juegos en HTML5 con una carga mínima de código. Ya que la mayoría de elementos se pueden crear usando herramientas visuales.

Amazon Lumberyard

Imagen 16: "Logo motor Amazon"

Desarrollado por Amazon, los describen como un motor gratuito para videojuegos AAA con un profundo nivel de integración con AWS (plataforma de servicios en la nube de Amazon) y twitch que incluye el código fuente completo.

Es un motor diseñado para juegos orientados al multijugador y que estén conectados al ámbito social (twitch, facebook etc.).

Actualmente está siendo usado por el juego "Star Citizen", el cual luce increíble, aunque aun se encuentra en fase de desarrollo.

Se podría decir que es un motor relativamente nuevo, pero como todo lo desarrollado por Amazon, seguro que no defrauda.

4.3.- Elección del motor

Debido a que este TFG será el primer videojuego que desarrolle, me he decidido finalmente debido a las siguientes razones:

1. Tiene una versión gratuita.
2. Una de las curvas de aprendizaje más sencillas.
3. Soporte completo para 3D.
4. Usa C# que es muy parecido a Java. Por lo tanto, no voy a necesitar aprender un lenguaje nuevo y totalmente diferente a los que ya conozco.
5. Ofrece bastantes facilidades a la hora de cambiar de plataforma.

4.4.- Plataformas móviles

Una de las decisiones que hubo que tomar, fue a qué tipo de dispositivo iba a centrarse el desarrollo de la versión móvil.

Android

Imagen 17: "Logo Android"

El sistema operativo Android ha sido desarrollado por Google, quien lo libero bajo licencia Apache, por lo tanto cualquier persona puede realizar una aplicación para esta plataforma.

La libertad de código permite adaptar Android a bastantes otros dispositivos además los teléfonos móviles, como por ejemplo Tabletas, relojes etc.

iOS

Imagen 18: "Logo iOS"

Desarrollada por Apple Inc. A diferencia de su competidor Android, no permite la instalación de su SO en hadware de terceros. Además, de que hay que pagar una cuota para publicar aplicaciones, son más difíciles de publicar, ya que hay que cumplir una mayor cantidad de requisitos respecto a Android.

4.5.- Elección de plataforma móvil

Se ha elegido desarrollar la aplicación móvil para Android, ya que ofrece muchísimas facilidades a la hora de instalar aplicaciones. Ya que en nuestro caso por ejemplo, lo que haremos será generar un fichero APK y Android nos da total libertad de instalar dichos ficheros aunque no sean oficiales.

Otras de las razones por las que se ha elegido este SO, es que en el caso de elegir iOS necesitaría un MAC para poder generar la aplicación desde Unity, ya que se requiere el IDLE Xcode que solo puedes usar en MAC o creando maquinas virtuales que simulen un mac.

5.- Análisis de la idea

5.1.- Juegos de referencia

En este apartado hablaremos de aquellos juegos en los que he basado algunos elementos de mi proyecto.

Door Kickers

Me encanto su estética. Además, hace un uso similar del área de las armas. Las mecánicas de movimiento no tienen nada que ver con mi juego.

Imagen 19: "Juego door kickers"

Mr. Shifty

La mecánica no tiene nada que ver, ya que este se trata de un juego cuerpo a cuerpo. Pero me dio la idea de crear grupos grandes de enemigos y tratar de crear un juego frenético donde el jugador se encuentre en tensión en todo momento.

Imagen 20: "Juego Mr. Shifty"

5.2.- Concepción de la idea

Se va a realizar un juego 3D del genero shooter con vista desde arriba para ordenador de sobremesa y Android. Donde se manejara un personaje con diferentes armas. El objetivo del juego será destruir portales y hordas de zombis.

La idea principal a la hora de diseñar los niveles será jugar con las luces creando columnas con lámparas que las proporcionen. Estas columnas se podrán destruir, por lo tanto puede llegar un punto que si nos quedamos sin columnas en una determinada zona, no podamos ver por donde se acercan los enemigos. Debido a que habrá pocas columnas a lo largo del nivel, he decidido añadir un casco con linterna para facilitar la tarea al jugador.

Los objetivos principales del nivel serán destruir zombis y el área desde donde aparecen. Una vez se destruyan todas estas áreas, el nivel terminara.

El jugador principal contara con las siguientes características:

1. Movimiento
2. Giro
3. Armas
4. Casco con linterna
5. Áreas de visión
6. Interfaz

A lo largo de los niveles nos encontraremos con 2 tipos de zombis:

1. Zombis normales
2. Zombis Alertadores

Además, se ha creado un nivel de entrenamiento, donde el jugador es prácticamente invencible y donde podrá matar tantos zombis como desee para así mejorar su habilidad o simplemente divertirse.

6.- Elementos de Unity

Para entender mejor esta memoria, vamos a explicar de forma sencilla algunos de los elementos de Unity que nombramos constantemente.

6.1.- Vistas

En Unity se usan pestañas denominadas vistas, las cuales nos permiten ver el juego desde diferentes perspectivas. Para realizar este TFG hemos usado las siguientes:

1. **Scene:** En esta vista podemos movernos libremente por todo nuestro juego. Ver todos los GameObject que lo componen y ver en vivo algunos de sus componentes, como por ejemplo ver donde están situadas las luces, los sonidos o ver hasta donde alcanza un determinado collider.

   En esta vista puedes modificar GameObject a tu gusto aunque el juego se encuentre en "play". Es decir, si por ejemplo quieres mover o cambiar de tamaño un GameObject lo puedes hacer sin problema.

Imagen 21: "Panel de Unity que permite mover, rotar, cambiar de tamaño, orbitar y hacer zoom sobre GameObjects o sobre la propia escena"

   Por último, esta vista nos muestra también la herramienta Gizmo, la cual nos permite modificar el Angulo de visión con un simple click.

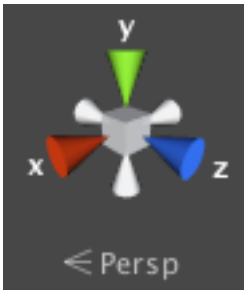


Imagen 22: "Gizmo"

2. **Game:** nos muestra el juego tal y como será visto por el jugador. Es una vista que no puede ser modificada en vivo.
3. **Inspector:** nos muestra todos los componentes de un GameObject. Es decir, todos los script que tiene relacionados, sus audios, efectos, colliders, rigibody etc.
4. **Lighting:** En mi caso, he usado esta vista únicamente para cambiar la intensidad de luz que hay sobre el mapa. De esta forma he podido crear zonas totalmente oscuras que impiden venir por donde vienen los zombis. Sin

embargo, esta vista permite muchas más opciones, sobre todo relacionadas con la iluminación general de nuestro proyecto.

5. **Navigation:** Vista que nos permite ver todas aquellas zonas navegables de nuestro mapa. Estas zonas serán navegables para todos aquellos GameObject que tengan un componente del tipo “Nav Mesh”.
6. **Animator:** Vista que nos permite ver las animaciones asociadas a un “Animator controller” y que transición siguen dichas animaciones. Más adelante hablaremos mejor de esta vista, ya que tiene una gran importancia a la hora de animar a un personaje.
7. **Hierarchy:** Contiene todos los GameObject que forman una escena.
8. **Project:** Contiene todas las carpetas que formas nuestro proyecto y nos permite navegar libremente por ellas.
9. **Console:** Vista que nos permite ver todos los errores y advertencias de nuestro proyecto.

6.2.- GameObjects

Son los elementos principales de nuestro proyecto. Finalmente nuestro proyecto será simplemente un conjunto de GameObject.

A un GameObject se le pueden asociar componentes para que así gane nuevas funcionalidades. Algunos de estos componentes son:

1. **Collider:** es un área alrededor de nuestro GameObject que nos permite a su vez detectar otros GameObject. Por ejemplo, para esta TFG se han usado sobre todo los collider para que los zombis sean capaces de detectar al jugador.
   Los collider además, pueden ser trigger o no. Cuando un collider es trigger evita que otro gameObject entre dentro del radio que ocupa.
2. **AudioSource:** Permite añadir un audio al gameObject.
3. **Script:** Permite añadir scripts al GameObject.
4. **Nav Mesh Agent:** Componente que nos permite mover un GameObject sin necesidad de hacerlo programando. Aunque si queremos aumentar la velocidad, aceleración u otros factores durante la ejecución, si que deberemos hacerlo mediante scripts.

Estos son algunos de los componentes que ofrece Unity. Sin embargo, el abanico de posibilidades que se nos ofrece es enorme y llegar a controlarlos todos requiere bastante práctica.

6.3.- Prefabs

Un prefabs es un GameObject que puede ser arrastrado a la pestaña hierarchy y de esta forma generaremos el GameObject en nuestra escena. Prácticamente cualquier GameObject de nuestra escena debería tener su propio prefab. De forma que deberíamos ser capaces de crear una nueva escena arrastrando únicamente prefabs hacia la pestaña hierarchy.

Un aspecto muy importante de los prefabs es que nos permiten crear nuevos GameObjects mediante el uso de Scripts. Basta con crear un script donde le asignemos GameObject públicos. Una vez realizado esto, en la pestaña inspector únicamente deberemos arrastrar el prefab que deseemos duplicar en el script.

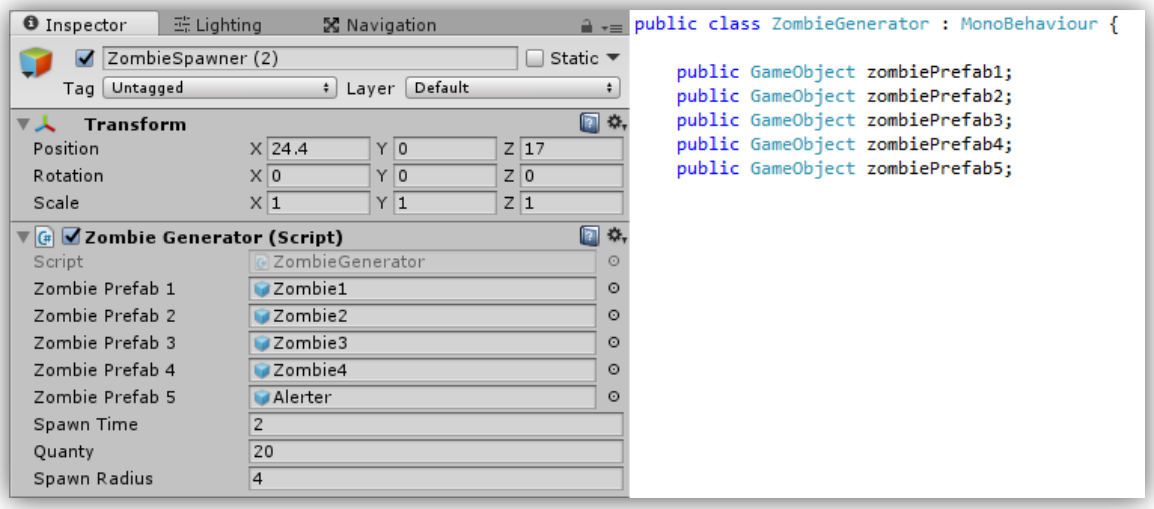


Imagen 23: "Ejemplo de cómo clonar GameObjects usando Prefabs"

7.- Características de los personajes

7.1.- Jugador Principal

Imagen 24: "Fotografía del jugador principal"

El jugador principal contara en todo momento con un área blanca a su alrededor, la cual indica el alcance de sus disparos independientemente de la dirección. También contara con un área azul, que nos indica el área que alcanzaran sus disparos al disparar en dicha dirección. Dependiendo del arma seleccionada esta área aumentara o menguara.

Finalmente, también cuenta con un casco con linterna. La cual puede ser encendida en cualquier momento por el jugador, sin embargo, se le ha añadido a la linterna una probabilidad de apagarse cada cierto tiempo para así aumentar la dificultad. La linterna se apagara automáticamente X segundos y se encenderá automáticamente pasado el tiempo de apagado.

Arsenal de armas

El arsenal de armas estas compuesto por:

1. **Pistola:** que apenas hace daño pero tiene un largo alcance.
2. **Escopeta:** lanza una gran cantidad de pequeños perdigones con direcciones aleatorias dentro de su área azul. Por separado hacen poco daño pero si varios tocan a un

enemigo hacen grandes destrozos. Esto convierte a la escopeta en un arma mortífera a corto alcance. Solo se puede disparar con ella una vez cada 2 segundos.

3. **Lanzallamas:** Hace una cantidad moderada de daño a todos los zombis que se encuentren dentro de su área azul. Solo se puede disparar con ella durante 10 segundos. Luego hasta que no pasen 3 segundos no podrás volver a usarla. Además, también es de gran utilidad para iluminar las zonas oscuras.

7.2.- Enemigos

Zombis normales

Imagen 25: "Fotografía de un zombi normal"

Tienen una inteligencia sencilla. Si entras dentro de su radio de detección pasaran de caminar sin rumbo a perseguirte a una velocidad moderada. Si durante dicha persecución el personaje principal logra salir del área (collider) de detección el zombi volverá a merodear sin un rumbo fijo a lo largo del mapa.

Si el zombi logra alcanzar al jugador le hará un daño bastante bajo y no podrá volver a realizar dicho daño hasta que pasen 3 segundos. A pesar de que el daño sea bastante bajo, hay que tener cuidado, ya que si se acerca una horda grande de enemigos cada uno de ellos si te alcanza te hará daño.

Zombis alertadores

Imagen 26: "Fotografía de un zombi alertador"

A diferencia del zombi normal, es un zombi que no hace ningún daño al jugador. Su función principal es detectar al jugador mediante 2 tipos de áreas (collider) que lo rodean. Un área emula la detección por sonido y tiene 10 de radio. Y la otra emula a la visión, permitiendo al zombi detectar al jugador a una gran distancia.

Una vez el zombi ha detectado al jugador, empezara a "gritar" y todos aquellos zombis que se encuentren dentro de cualquiera de sus dos áreas serán enviados a la posición del jugador.

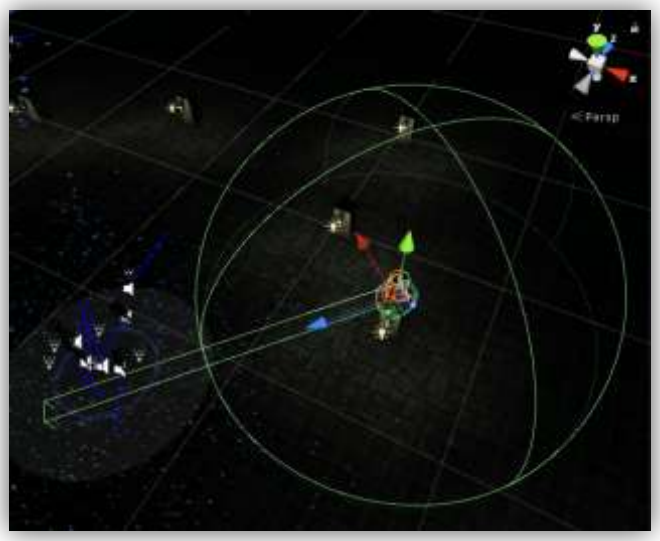


Imagen 27: "Colliders de un zombi alertador"

8.- Simulación de multitudes

La simulación de multitudes es el proceso de simular el movimiento de un gran número de objetos o personajes, apareciendo ocasionalmente por ejemplo en gráficos 3D por computadora para películas. Mientras se simulan estas multitudes, se toma en cuenta la interacción del comportamiento humano observado, para replicar la conducta colectiva.

La necesidad de una simulación de multitudes surge cuando una escena necesita más personajes de los que pueden ser animados de forma práctica usando sistemas convencionales, tales como esqueletos/huesos. Simular multitudes ofrece las ventajas de ser rentable así como permitir el control total de cada personaje o agente simulado.

Los actuales movimientos e interacciones de la multitud son típicamente hechos en una de dos formas:

1. **Movimiento de partículas:** Los personajes son adjuntados a partículas puntuales, que son entonces animados mediante la simulación del viento, gravedad, atracciones, y colisiones. El método de partículas es usualmente barato de implementar, y puede ser hecho en la mayoría de las paqueterías de software 3D. Sin embargo, el método no es muy realista debido a que es difícil dirigir las entidades individuales cuando es necesario, y porque el movimiento está generalmente limitado a una superficie plana.
2. **IA de multitud:** A las entidades, llamadas también agentes, se les da inteligencia artificial, que guía a las entidades basadas en una o más funciones tales como vista, audición, emoción básica, nivel de energía, nivel de agresividad, etc. Se les otorga a las entidades de metas y entonces interactúan entre sí como lo harían los miembros de una multitud real. Son en ocasiones programadas para responder a cambios en el ambiente, permitiéndoseles escalar colinas, saltar sobre hoyos, subir escaleras, etc. Este sistema es mucho más realista que el movimiento de partículas, pero es muy costoso de programar e implementar.

   Los ejemplos más notables de simulación de IA pueden ser vistos en las películas de New Line Cinema El Señor de los Anillos, donde ejércitos de IA de muchos miles de soldados pelearon entre sí. La simulación de multitud fue hecha utilizando el software MASSIVE de Weta Digital.

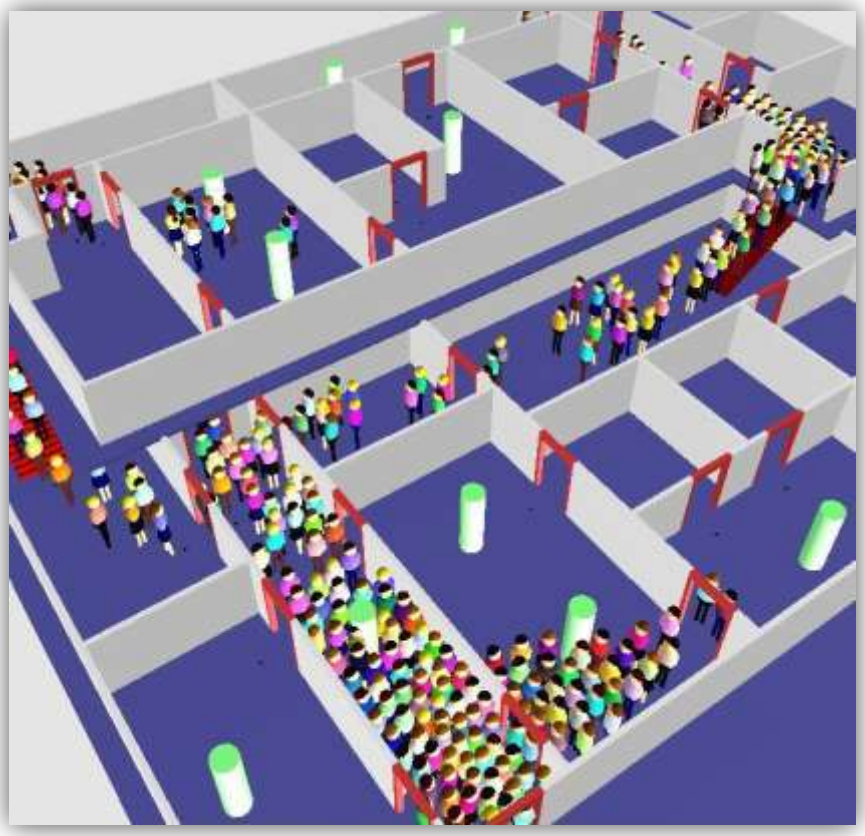


Imagen 28: "Ejemplo de simulación de multitudes"

8.1.- Ataque usando la simulación de multitudes

Para aumentar la dificultad del juego, se ha desarrollado un script donde se hace uso de la IA de multitudes para atacar al jugador en grupo. Estos grupos estarán compuestos únicamente de zombis normales.

El script se ha añadido como componente a un objeto vacio con un determinado radio de alcance. Dicho objeto se ha puesto cerca de un lugar donde aparecen los zombis.

Su funcionamiento consiste en detectar los zombis que se encuentran dentro de su área y almacenarlos dentro de una lista. Una vez dicha lista tiene un tamaño determinado, comienza la táctica.

La táctica consiste en enviar a los zombis de la lista desde su posición hacia las esquinas del mapa. Se enviara una cantidad aleatoria de zombis de la lista a cada esquina (no tienen porque enviarse a todas las esquinas, a lo mejor a alguna no se mandan, ya que se eligen las esquinas de forma aleatoria).

```
public class MapSize : MonoBehaviour {

    public Renderer mapSize;

    void Awake () {
        mapSize = gameObject.GetComponent<Renderer>();
        Debug.Log( mapSize.bounds.max );
        Debug.Log( mapSize.bounds.min );
    }
}
```

Imagen 29: "Código para detectar las esquinas del mapa"

Una vez han llegado un mínimo de zombis a las esquinas, se les informa de la posición del jugador y se les asigna una velocidad dependiendo de la distancia a la que se encuentren del jugador. De esta forma, cuanto más cerca te encuentres del jugador, menor será la velocidad y cuanto más lejos la velocidad será mayor. Logrando así, que todos los zombis lleguen prácticamente a la vez al jugador desde diferentes flancos.

Imagen 30: "Visualización de cómo llegan los zombis desde 4 direcciones diferentes"

La velocidad a la que se mueven los zombis hacia el jugador varía entre 0 y 5. Esto provoca que la técnica no sea perfecta, ya que si el jugador se mueve hacia una esquina sin parar, por muy rápido que se muevan el resto de zombis hacia el jugador nunca lograran alcanzarlo antes de que el jugador alcance al grupo que se encuentra en la esquina hacia la que se dirige.

Este último caso, podemos verlo en los extras del juego, más concretamente los videos de “dynamic crowd simulation”. Donde podremos observar que el jugador se mueve continuamente hacia diferentes direcciones, por lo tanto los zombis no llegan simultáneamente al jugador. Esto se podría solucionar haciendo que los zombis corrieran por ejemplo 4 veces más que el propio jugador o teletransportandolos directamente a X radio de distancia del jugador. Sin embargo, haciendo esto, el juego perdería realismo.

9.- Desarrollo del proyecto

Antes de empezar a crear cualquier tipo de contenido visual me he centrado en crear unas bases solidas para mi juego. Donde se plasman las mecánicas base del juego. Estas mecánicas tienen que ver con:

- Movimiento del personaje y enemigos
- Crear un objeto que añadiese enemigos al nivel de forma automática
- Crear los estados de los enemigos
- Crear las armas

Una vez se han desarrollado estas mecánicas, ya podemos empezar con otras tareas como añadir texturas, crear el mapa, añadir efectos etc.

9.1.- Mecánicas básicas

Movimiento y giro

Como primer objetivo de esta fase, nos hemos centrado en el movimiento del personaje y de los enemigos. Para ello hemos usa varios GameObject con forma de cubo y les hemos asignado un script de movimiento.

En los primero prototipos el movimiento del personaje se configuró para realizarse con "WASD" y el giro con "QE". Sin embargo, se hacía extraño girar sobre si mismo haciendo uso del teclado. Por lo que se decidió implementarlo haciendo uso del ratón. Para implementar estos giros nos hemos basado en la posición que toma constantemente el ratón en el mapa y la posición en la que se encuentra el jugador.

Respecto al movimiento de los enemigos, en un principio se diseñó un script que los movía. De hecho iba perfecto, siempre tomaba el camino más corto hasta el jugador. Sin embargo, cuando se introducían obstáculos no eran capaces de esquivarlos ni de tomar la dirección más corta hacia el jugador a la vez que esquivaban el obstáculo.

Para solucionar estos problemas, se optó por hacer uso del componente "Nav Mesh Agent" que proporciona Unity. Este componente sigue el algoritmo A* para mover a los GameObjects que lo tengan asignado. Y además, durante estos movimientos es capaz de detectar los obstáculos.

¿Cómo funciona el algoritmo A*?

Antes de empezar a explicar el algoritmo A* vamos a explicar los 2 algoritmos de búsquedas en grafos en los que se basa:

- **Ramificación y acotación:** está compuesto por una lista ABIERTA de trayectorias parciales que contienen los nodos recorridos, ordenadas de menor a mayor costo acumulado, de manera que la primera trayectoria de la lista es siempre la de menor coste acumulado.

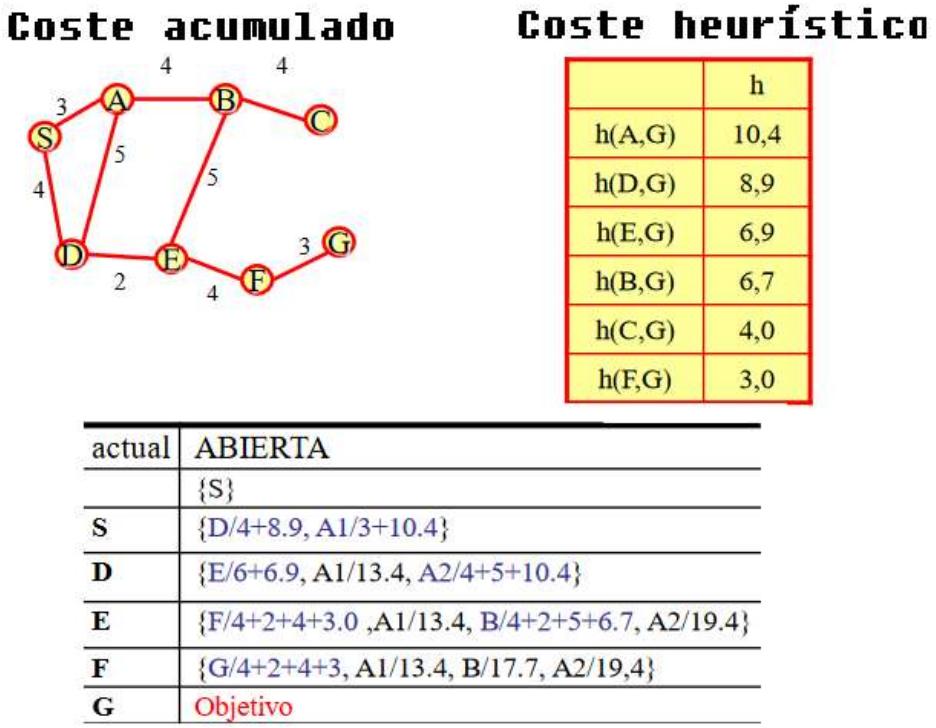
La idea es que la primera trayectoria de la lista ABIERTA genere, mientras sea posible su expansión, nuevas trayectorias por ramificación de sus hijos.

- **Ramificación y acotación con subestimación:** Mejora el algoritmo de ramificación y acotación, ya que no sólo considera el camino hasta alcanzar un estado, sino también las expectativas acerca de lo próximo que está ese estado del estado objetivo.

  Se hace uso de una función de coste total estimado acumulado para cada nodo intermedio n en el proceso de exploración:

  F*(n) = g(n) + h*(n)

  - **G(n):** coste real acumulado (el mismo que usamos con ramificación y acotación)
  - **H*(n):** coste heurístico del estado actual al estado objetivo



	h
h(A,G)	10,4
h(D,G)	8,9
h(E,G)	6,9
h(B,G)	6,7
h(C,G)	4,0
h(F,G)	3,0

actual	ABIERTA
	{S}
S	{D/4+8.9, A1/3+10.4}
D	{E/6+6.9, A1/13.4, A2/4+5+10.4}
E	{F/4+2+4+3.0 ,A1/13.4, B/4+2+5+6.7, A2/19.4}
F	{G/4+2+4+3, A1/13.4, B/17.7, A2/19,4}
G	Objetivo

Imagen 31: "Problema ejemplo de ramificación y acotación con subestimación"

El algoritmo A* es muy similar a ramificación y acotación con subestimación, pero además añade una lista cerrada en la que va añadiendo rutas que se han descartado por ciertos motivos. Esta lista cerrada permite eliminar rutas de la lista abierta que no sean viables, reduciendo considerablemente el número de nodos expandidos.

¿Cómo detecta las colisiones un Nav mesh agent?

Simplemente marcando el GameObject como estático. Una vez realizado, unity se encarga de detectar que zonas del mapa son navegables y cuáles no.

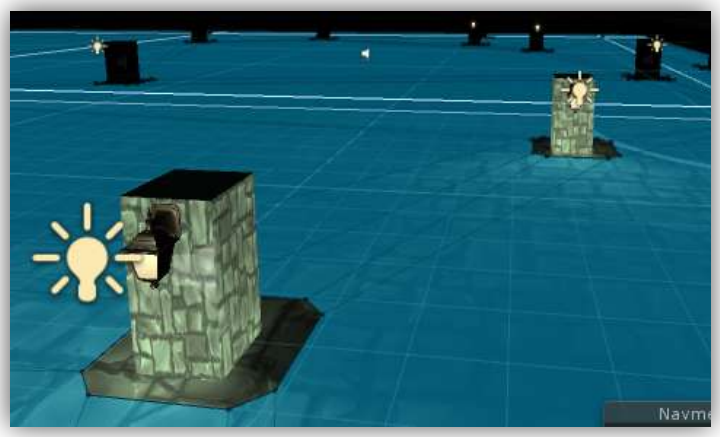


Imagen 32: "Zonas navegables del mapa"

Además, también podremos indicar algunas características de estas zonas no navegables, como su radio, altura etc.

Añadiendo enemigos al nivel

Para ello se ha creado una clase que se la ha asignado a un gameObject y cuya función consiste en cada X tiempo reproducir un zombi en una posición aleatoria dentro de una determinada área. El tipo de zombi creado se hace al azar usando prefabs, por lo tanto, puede ser que se cree tanto un zombi normal como un zombi alertador.

A este gameObject se le puede modificar varios valores desde el entorno grafico, como por ejemplo la cantidad máxima de zombis que puede crear, el tiempo de creación entre zombis y el área aleatoria máxima en la que puede crearse el zombi.

Estados de los enemigos

El estado de los enemigos se ha creado mediante "coroutines", que nos permiten hacer ciertas acciones mediante un bucle infinito. Todos los enemigos tienen en común dos estados:

- **Wander:** es el estado de vagabundeo, se usa para que el zombi se mueva de forma aleatoria sin rumbo mientras no detecte al jugador.
- **Chase:** en este estado se entra cuando se detecta al jugador y dependiendo del zombi hará una acción u otra.
- **CrowdAttack:** este estado no es común para todos los zombis y solo lo tienen los zombis normales para coordinarse mientras realizan un ataque en grupo.

```
public enum state{
    Chase,
    Wander,
    CrowdAttack,
}

IEnumerator stateMachine(){
    while( true ){
        yield return StartCoroutine( currentState.ToString() );
    }
}
```

Imagen 33: "Estados de los zombis y como cambiar entre ellos"

Para cambiar de un estado a otro, en casi todos los casos se han usado colliders. Mientras se detecte al jugador dentro del collider ejecuta X estado. Si el jugador sale del collider ejecuta Y estado.

Armas

Para crear las armas, se ha creado la clase abstracta "Weapon" que define elementos comunes para todas las clases de armas que la extienden. Como por ejemplo el daño de los proyectiles, área de alcance del arma, tiempo de reutilización etc.

El cambio entre armas se realiza con la rueda del ratón y mediante un script lo que se hace es desactivar todas las armas que no sean la objetivo. De esta forma eliminamos todos sus subelementos como las áreas blancas y azules.

Características de las armas:

- Pistola
  - Cantidad de proyectiles: 1
  - Daño por proyectil: 2
  - Tiempo de reutilización: 0.6s
  - Rango del arma: Vector2(20, 20)
- Escopeta:
  - Cantidad de proyectiles: 35
  - Daño por proyectil: 0.4
  - Tiempo de reutilización: 2s
  - Rango del arma: Vector2(10,10)
- Lanzallamas
  - Cantidad de proyectiles: 0, hace daño a todo aquel que entre dentro de su rango azul
  - Tiempo de uso: 10s
  - Tiempo de reutilización: 3s
  - Rango del arma: Vector2(10,10)

En un principio, la idea era crear un juego frenético de disparos. Es decir, un juego donde no te tuvieras que preocupar por elementos como la munición o que las armas no tuviesen tiempo de reutilización. Sin embargo, tras varias pruebas, se llego a la conclusión de que con esa mecánica el juego sería muy sencillo, por ello se decidió añadir elementos como el tiempo de reutilización.

9.2.- Creación de los niveles

Una vez se sentaron las bases del juego, se procedió a la creación de los niveles. La idea a la hora de crear los niveles consiste en jugar con la iluminación. De forma que un nivel con mucha iluminación será sencillo, ya que en todo momento ves por donde vienen los enemigos, mientras que un nivel poco iluminado es más complicado, ya que no ves por donde se acercan los enemigos y si encima coincide el momento que no hay luz con un ataque de multitudes puede significar fácilmente la muerte del jugador. Esta iluminación se proporciona mediante columnas (obstáculos) con luces, las cuales pueden ser destruidas con disparos.

En un principio la idea era crear dos niveles, uno con mucha iluminación y otro con poca, sin embargo, una vez realizados ambos, el primer nivel casi no tenia dificultad, ya que no solo tenía mucha iluminación, sino que al tratarse de un tutorial no se implemento el ataque de multitudes. Por lo tanto, se decidió que en lugar de crear dos niveles, se creara solo 1 y otro nivel de prácticas donde el jugador tuviese vida casi infinita y los enemigos no parecen de aparecer. Además, se deciden añadir trampas en el escenario, que son un arma de doble filo. Por un lado, vienen muy bien para matar hordas de zombis, pero por otro, como una trampa sea activada por un zombi y dañe al jugador, perderá una enorme cantidad de vida. Existen dos tipos de trampas:

1. **Trampas con una cruz blanca:** emiten áreas de fuego alrededor del gameObject que la pisa.
2. **Trampas con área roja:** emiten áreas de fuego en la propia área roja.

La creación de los mapas se hace de forma automática mediante un script que está asignado a un gameObject. A dicho gameObject le puedes indicar el número de obstáculos (columnas que proporcionan luz) y el tamaño del mapa. Tras ello, se generara un mapa aleatorio que contenga todos los elementos indicados. Además, se pueden cambiar las combinaciones de los obstáculos y la separación de las baldosas que componen el piso. Por lo tanto, tenemos una herramienta que prácticamente nos puede generar niveles infinitos.

Otro punto muy importante de este script, es que podemos modificar el espaciado que hay entre obstáculos. De forma que si ponemos un espaciado de 0, las columnas se podrán juntar unas con otras, formando caminos. Sin embargo, para este juego se ha decidido poner un espaciado bastante amplio entre las columnas, para así dar libertad de movimiento al jugador y que no resulte agobiante tener que seguir un determinado camino o tener que estar destruyendo constantemente columnas para abrirte paso a otras zonas.

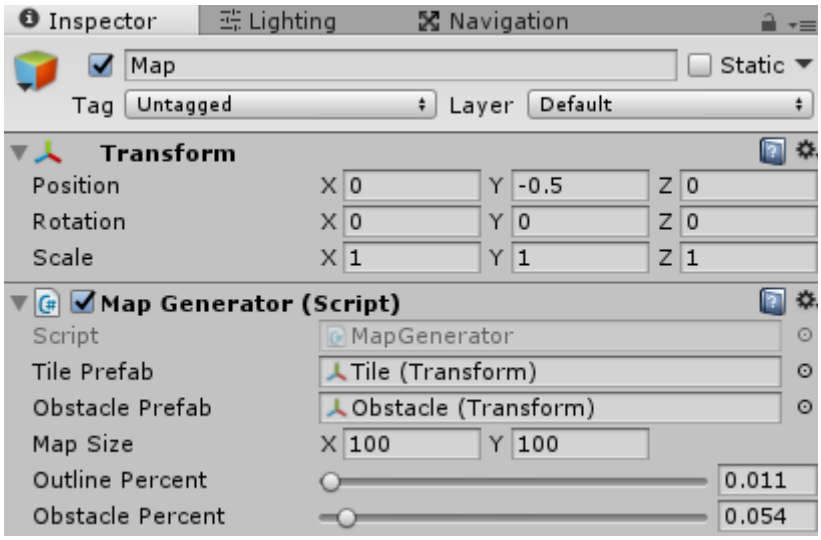


Imagen 34: "GameObject que contiene el scrip que genera mapas"

Por último, recalcar que la iluminación en este juego juega un factor muy importante, no solo por la dificultad que comentábamos anteriormente, sino porque al tratarse de un juego de miedo tiene un papel fundamental.

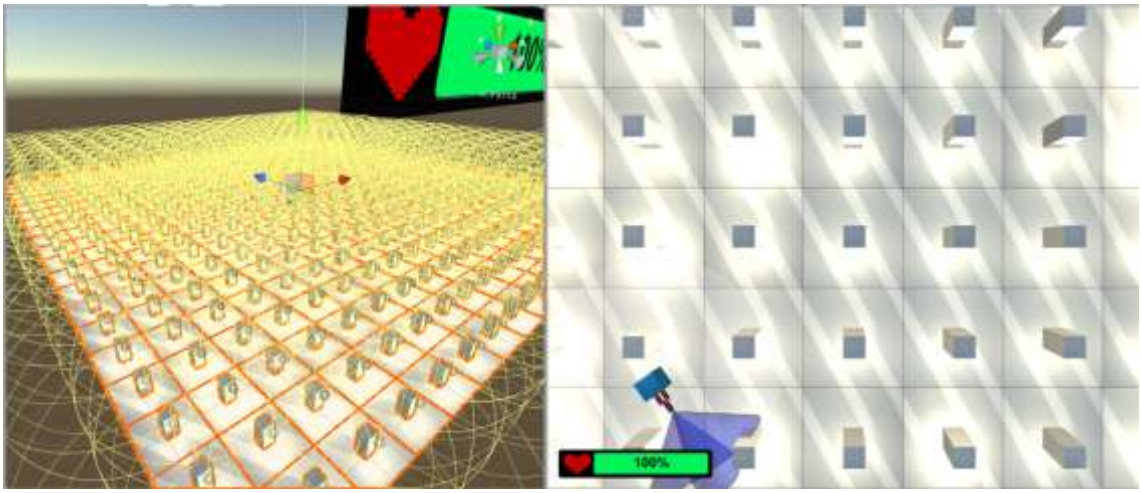


Imagen 35: "Ejemplo de un nivel totalmente iluminado"

9.3.- Efectos de partículas

En un juego 3D, la mayoría de los personajes, accesorios y elementos de escenario son representados como meshes, mientras que un juego 2D utiliza sprites para estos propósitos. Los Meshes y Sprites son la manera ideal de representar objetos "sólidos" con una figura bien definida. Hay otras entidades en los juegos, no obstante, estas son fluidas e intangibles en la naturaleza y, por consiguiente son difícil de representar utilizando Meshes o Sprites. Para efectos como líquidos en movimiento, humo, nubes, llamas y hechizos mágicos, hay un enfoque diferente a los gráficos conocidos como **particle systems** que pueden ser utilizados para capturar la fluidez inherente y la energía.

Los efectos de partículas son increíbles en unity. Podemos crear prácticamente cualquier efecto a partir de la nada. Y si encima usamos imágenes como materiales, nos dan como resultados unos efectos tremendamente realistas y "sencillos" de crear (aunque requieren bastante practica). Podemos simular desde una simple caída de muchos cubos hasta la creación de un tornado de fuego, mares etc.

En esta sección enseñaremos algunos de los efectos de partículas creados para el juego.

Fuego

Efecto de partículas usado en la pantalla principal del juego (braceros y fuego generado al pulsar el botón) y en el lanzallamas. Cada uno con sus características especificas. Como la gravedad para el caso del brasero, la curva de crecimiento para el botón etc.

Imagen 36: "Diferentes efectos de fuego del juego"

Escopeta

Creación de un haz de luz cada vez que disparamos

Imagen 37: "Efecto de partículas al disparar con la escopeta"

Efecto de muerte del jugador y columnas

Debido a que las columnas son rectángulas y el jugador es un cubo, a la hora de morir o destruirse, lo que se ha hecho ha sido crear un efecto de partículas en el que pequeños cubos caen en la posición donde se ha destruido el objeto, dando la sensación de derrumbamiento o destrucción.

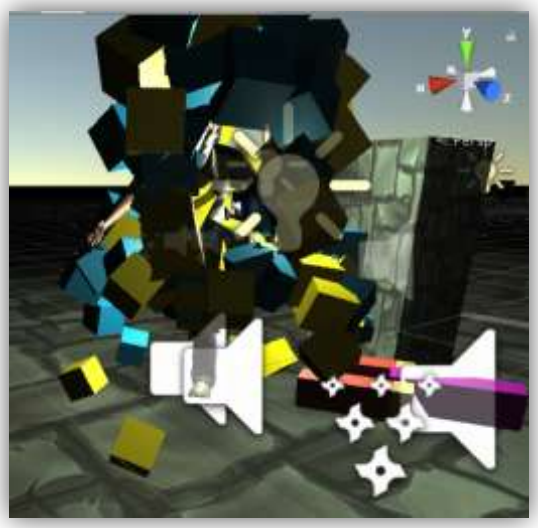


Imagen 38: "Efecto de mini cubos cayendo para simular la muerte del jugador"

Portales

Los portales están compuestos por 3 efectos de partículas:

1. Un efecto está dedicado a mover partículas de fuera hacia dentro en la zona del agujero del portal.
2. Otro efecto mueve partículas alrededor del portal.
3. El ultimo efecto consiste en enviar partículas hacia la cámara

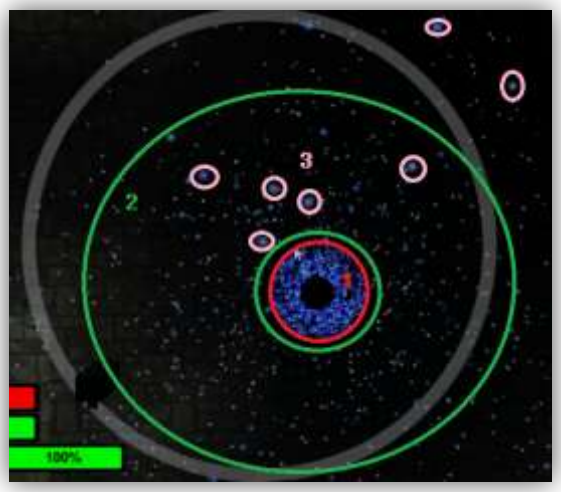


Imagen 39: "Partículas de los portales"

Sangre

Se han creado dos efectos de sangre:

1. Cuando una bala impacta con el enemigo aparecen pequeños charcos de sangre
2. Cuando un enemigo se arrastra deja un rastro continuo de sangre que varía en tamaño según va avanzando.

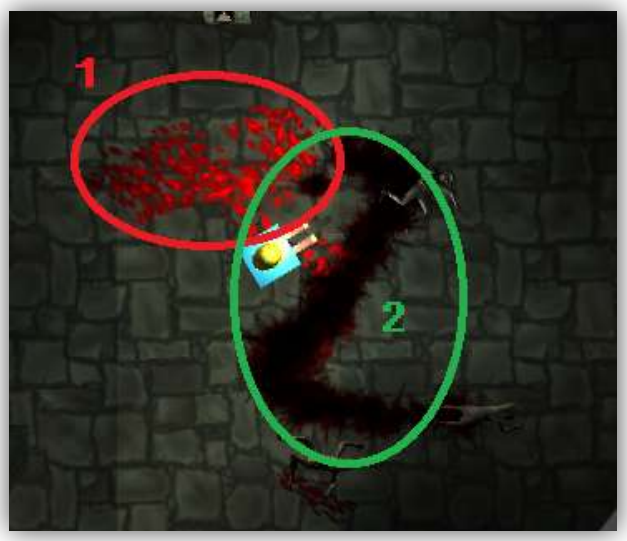


Imagen 40: "Efectos de sangre"

9.4.- Efectos de sonidos

En esta fase del proyecto me he dedicado a añadir efectos de sonidos a los gameObjects de la escena.

El control de estos elementos se puede hacer tanto mediante Unity como mediante código. Sin embargo, Unity te permite un manejo muy simple. Si quieres que un sonido se ejecute en un determinado momento y pare en otro momento, deberás hacerlo mediante código.

Algunos de los elementos de sonido que hemos añadido en esta fase, han sido:

- Sonido
  - Zombis
    - Para cada uno de sus estados tiene un sonido diferente
  - Jugador
    - Sonidos para cuando recibe daño, cuando le queda poca vida, cuando enciendes la linterna de su casco etc.
  - Columnas
    - Sonido de luz fundida y sonido para cuando las destruyes.
  - Armas
  - Ambiente del mapa
  - Portales

9.5.- Añadiendo las texturas

En este punto del proyecto, ya tenemos el movimiento implementado, los enemigos, las mecánicas, las armas, los niveles etc. Sin embargo, todo estaba representado de forma muy básica. Prácticamente todos los elementos de nuestro juego eran cubos formados por mas cubos más pequeños o por rectángulos. Así que ya era la hora de añadir las texturas y darle algo de realismo a todos nuestros elementos.

En nuestro caso, se han usado texturas del formato “fbx” y Unity no ha puesto ningún impedimento a la hora de asignarlas a los GameObjects.

Añadir texturas es bastante simple en Unity, simplemente cogemos nuestra textura y la arrastramos a la vista “Project”. Una vez realizado esto, Unity creara un prefab de dicha textura, lo que nos permitirá añadirla a la vista “Hierarchy” sin ningún problema.

En nuestro caso, para que todos los zombis normales no fuesen iguales, lo que hemos hecho ha sido crear 6 prefab con texturas diferentes, pero realmente todos tienen los mismos componentes que un zombi normal. Por lo tanto, aunque veamos que un gameObject es un policía zombi y otro gameObject es una enfermera zombi, realmente todos tienen el mismo comportamiento (el de un zombi normal).

Una vez hemos añadido las texturas a todos los elementos, tocaba animar todas aquellas que tenían que ver con los enemigos, ya que si no nuestros enemigos estarían todo el rato en una misma posición y no darían sensación de movimiento.

Imagen 41: "Ejemplo de textura de zombi"

9.6.- Añadiendo las animaciones

Para asignar animaciones a un GameObject en Unity necesitamos crear un componente del tipo "Animator" y asignarle un "Animator controller". Una vez creado, este "animator controller" nos permite el acceso a una maquina de estados. En dicha maquina iremos creando estados y asignándoles animaciones. Además, también podremos indicar a que estado puede cambiar. Permitiendo así, que al cambiar de estado se produzca una nueva animación.

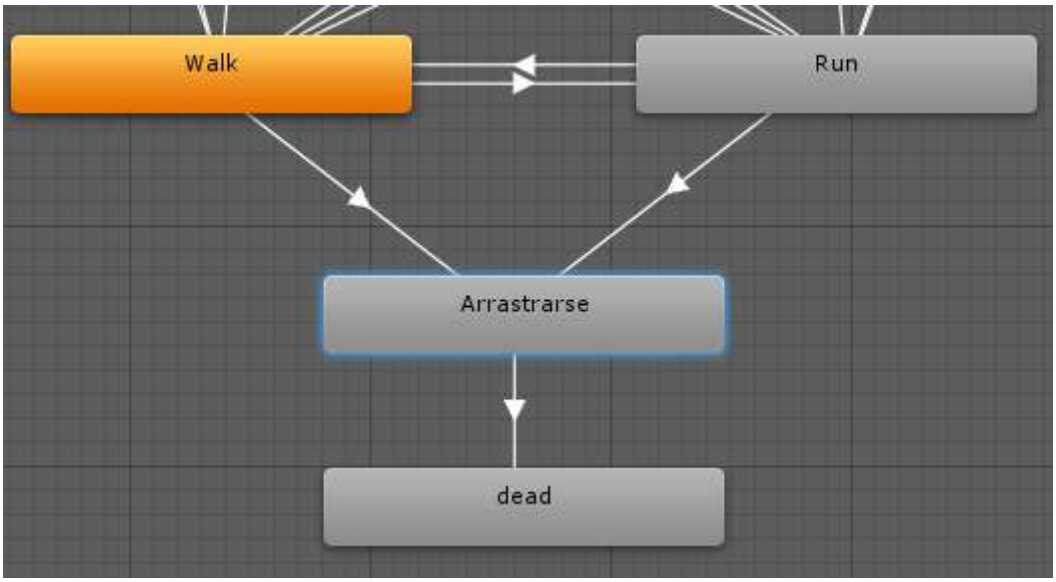


Imagen 42: "En este ejemplo tenemos 4 estados y como observamos se puede viajar de unos a otros. Por ejemplo, del estado arrastrarse puedes ir al estado dead"

Para viajar entre estados usaremos “variables” y las asignaremos a los estados. Por ejemplo para movernos del estado “arrastrarse” al estado “dead” es necesario que la variable “dead” este a “true”. Además, Unity también nos permite indicar el tiempo que va a tardar la animación de un estado en cambiar a la animación del estado al que se mueve. De forma que podemos cuando se active la variable podemos hacer que dicha transición ocurra en 2 segundos o en el tiempo que deseemos.

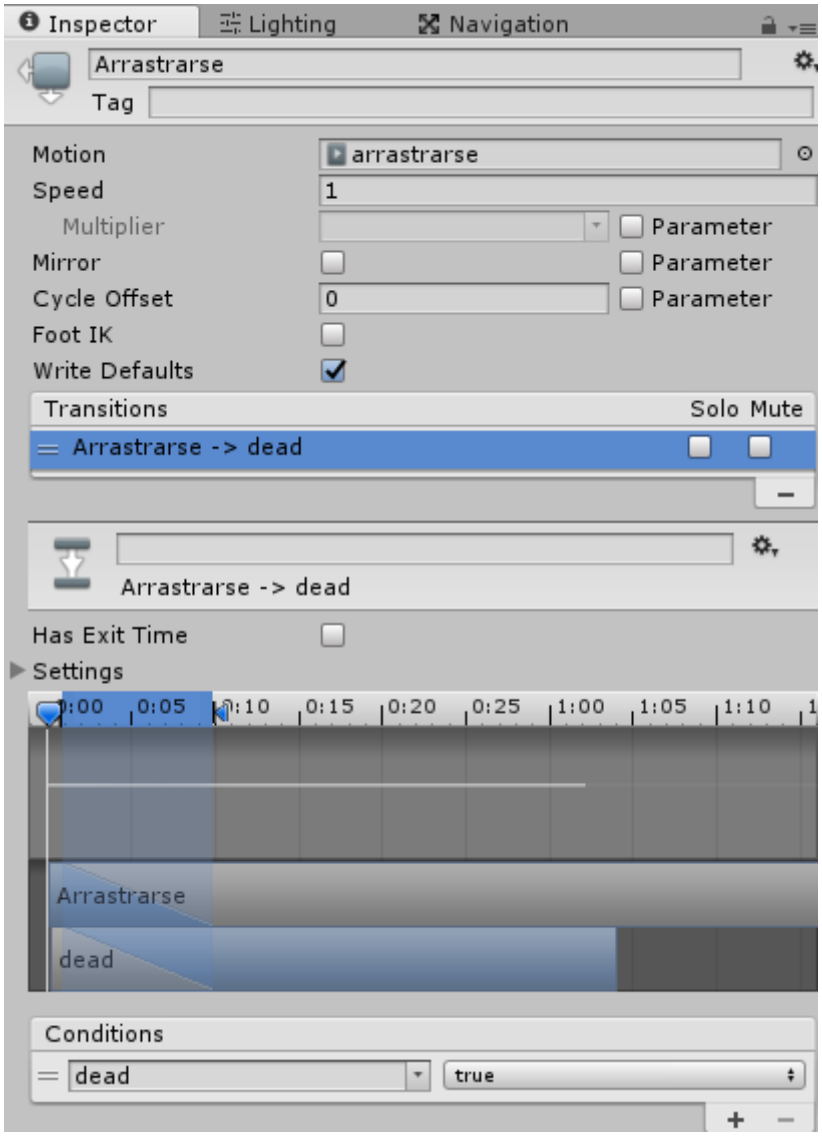


Imagen 43: “*Para el estado arrastrarse ejecutaremos la animación arrastrarse a velocidad 1. Desde este estado será posible cambiar únicamente al estado dead siempre y cuando la variable dead este a true. Además, dicha transición de animaciones se hará en 0:08 segundos*”

¿Cómo activar estas variables?

Lo primero que debemos hacer es crear una variable del tipo "Animator" y asignarle el componente "Animator" de nuestro gameObject. A partir de ahí, todas las variables que hemos creado mediante el IDE Unity estarán disponibles en la variable de nuestro Script y nos permitirá activar o desactivar las variables de Unity de la siguiente forma:

```
if( gameObject.name.ToString().Contains("Zombie") && health > 0 ) {
    if( UnityEngine.Random.Range( 0, 6 ) == 5 ){
        GetComponent<Zombie>().animations.SetBool( "arrastrarse", true );
        GetComponent<Zombie>().speed = 0.7f;
```

Imagen 44: "En este caso vemos que si tras tirar un random, el valor que sale es un 5, se activara la variable arrastrarse y la pondremos a true. Además, al haber activado la animación de arrastrarse disminuimos la velocidad del zombie. Ya que una persona que se arrastra va más lento de lo normal"

9.7.- Creación de la interfaz

Para crear interfaces en Unity, haremos uso de un Gameobject "**Canvas**".

Imagen 45: "Interfaz del jugador"

¿Qué es Canvas?

El Canvas es el área donde todos los elementos UI deben estar. El Canvas es un Game Object con un componente Canvas en él, y todos los elementos UI deben ser hijos de dicho Canvas.

Creando un nuevo elemento UI, tal como una Image (imagen) utilizando el menú GameObject > UI > Image, automáticamente crea un Canvas si ya no hay uno en la escena ya. El elemento UI es creado como un hijo de este Canvas.

El área Canvas es mostrado como un rectángulo en la Vista de Escena. Esto lo hace fácil posicionar los elementos UI si necesitar tener una Vista de Juego todo el tiempo.

A un elemento Canvas se le pueden añadir otros GameObjects que a su vez contengan componentes del tipo “imagen”, “texto”, “botón” etc.

Por ejemplo, para crear el botón del menú inicial (el que al pulsarlo activa una trampa de fuego que mata a los zombis), lo que hemos hecho ha sido crear un canvas, que dentro contiene un GameObject con un componente imagen (la imagen del botón) y 2 componentes de textos, que contienen los textos que encontramos a los lados del botón. Además, el gameObject que contiene el componente imagen contiene otro componente del tipo “Button”. Este componente nos permite indicar el método que se va a ejecutar al presionar sobre la imagen.

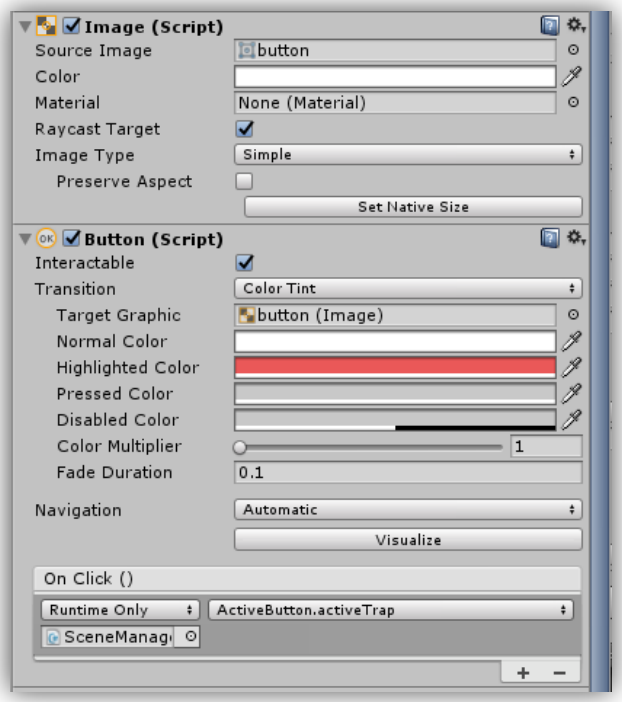


Imagen 46: “Ejemplo del componente button”

9.8.- Creación de las escenas

¿Qué es una escena?

Las escenas en unity contienen todos los gameObject de una determina fase de nuestro juego. Las escenas son individuales y hay que pensar en ellas como en niveles únicos. Por ejemplo tenemos las escena 1 que corresponde al nivel 1 de nuestro juego y tiene X gameObject que tienen que ver únicamente con dicha escena.

En las escenas cobran gran importancia los prefab, ya que al tener creados prefabs los podemos añadir a cualquier escena simplemente arrastrándolos desde la vista Project.

Movimiento entre escenas

Otra característica muy importante de las escenas, es que nos podemos mover entre ellas. De esta forma mediante código podemos decir por ejemplo: cuando el jugador complete X objetivos activa la escena "nivel final". Y básicamente así es como en Unity se cambia de pantallas o niveles.

```
public void activeLevel2() {
    Application.LoadLevel( "Level2" );
}
```

Imagen 47: "Activar la escena llamada Level2"

Escenas creadas para este juego

- Pantalla inicial
- Pantalla de extras
- Pantalla de game over
- Pantalla de fin del juego
- Nivel 1 del juego
- Nivel de entrenamiento

Un pequeño extra

Antes de empezar este TFG, recuerdo que cuando jugaba a "World of Warcraft" había una pantalla de inicio donde indicabas tu nombre de usuario y contraseña para entrar al juego. En dicha pantalla había un efecto de un portal en movimiento y recuerdo que yo me quedaba pensando: ¿Qué será ese portal de ahí detrás? ¿Un gif?

Ahora puedo decir casi con total certeza, que se trata de una escena donde el portal es un efecto de partículas y que cuando pulsas en el botón "jugar" lo que se hace es cambiar a otra escena.

Imagen 48: "Pantalla de login del juego World Of Warcraft The Burning Crusade"

9.9.- Adaptación a Android

Para hacer la versión móvil de nuestro juego ha habido que cambiar unos cuantos elementos, sobre todo los relacionados con los controles.

Como es lógico, Android no es capaz de detectar los mismos controles que usamos para la versión de ordenador de sobremesa. Por ello hubo que realizar una nueva interfaz que permitiese el movimiento y los cambios de armas, así como disparar.

En un principio, se desarrollo únicamente un joystick para que el personaje se pudiese mover. El tema del giro se pensó hacer con el dedo, es decir, a donde pulses con el dedo en la pantalla es a donde girara el personaje. Sin embargo, esto no fue posible, ya que al tener dos dedos en la pantalla simultáneamente (uno para moverte y otro para disparar), Unity nos devolvía la media de las posiciones. Así que a la hora de indicar la dirección hacia donde debía girar usaba dicha media y no se posicionaba donde el jugador picaba con el dedo.

Para solucionar el problema del movimiento, se han creado 2 joystick que emulan el giro y el movimiento del personaje y en lugar de hacer del click del ratón se han usado las herramientas que proporciona Unity para la detección de dedos en la pantalla. Dando como resultado que se puedan usar los 2 joystick sin problemas y cada uno sea independiente del otro.

Hubo que realizar otros cambios en la interfaz, debido a que en PC podíamos asignar teclas para cualquier elemento sin problema, pero en móvil apenas tenemos teclas que presionar. Los demás cambios realizados fueron:

- Las armas ahora se cambian mediante un botón rojo que se inserto en la interfaz
- Para disparar hay que hacer doble click sobre la pantalla
- Creación de un botón “auto” para disparar de forma automática sin necesidad de tener que estar haciendo todo el rato doble click sobre la pantalla
- La linterna del casco se enciende ahora haciendo click sobre su icono de la interfaz
- Para pausar el juego hay que apretar en el botón de “retroceso”

Otro problema que surgió durante la adaptación a Android, es que la cantidad de enemigos en el mapa no es soportable para un móvil. En PC por ejemplo, hay puntos que en nuestro juego hay simultáneamente ~100 zombis deambulando por el mapa. Esta cantidad de zombis no puede ser soportada por un móvil ya que no cuenta con un procesador, ram y grafica tan potente como un PC. Así que hubo que disminuir la cantidad de zombis que aparecen a través de los portales así como los zombis que atacan mediante la simulación de multitudes.

Finalmente, en la versión de móvil se pueden emular todos los movimientos, sin embargo, **no es tan cómodo de manejar ni preciso como en PC**, esto provoca un aumento en

la dificultad. Así que además de disminuir el número de zombis como indicamos anteriormente, también se duplico la vida del personaje principal y se eliminaron las trampas, ya que estas últimas requieren de un movimiento muy preciso para ser esquivadas y si fallas restan mucha salud.

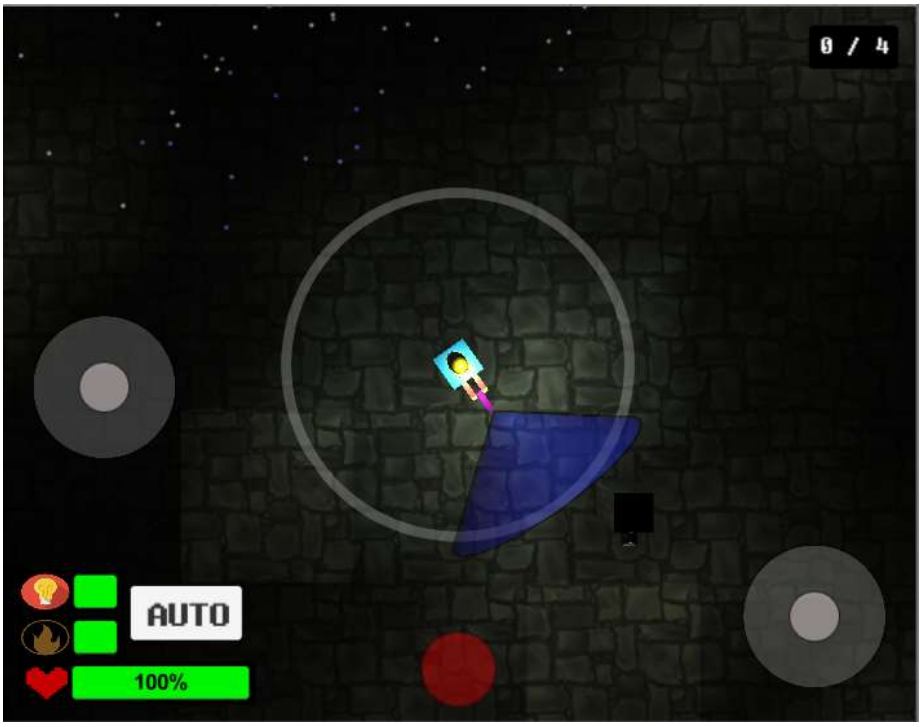


Imagen 49: "Interfaz para dispositivos moviles"

10.- Pruebas y corrección

Las pruebas han sido un elemento utilizado durante todo el desarrollo. Cada vez que se añadía una nueva funcionalidad o características se procedía a probar que funcionase correctamente. Además, también se comprobaban de forma breve que todas las funcionalidades anteriores también seguían funcionando sin problema.

Durante el desarrollo también se ha contado con la ayuda de familiares y amigos para que probasen el juego y diesen opiniones o encontrasen bug.

Muchas de estas opiniones tenían que ver con los controles, ya que el tratarse de personas de unos ~50 años no han jugado a videojuegos a lo largo de su vida, así que no suelen conocer los controles típicos de un juego ("WASD", espacio etc.). Este problema finalmente se soluciono indicando en el menú pause las teclas de uso.

También hubieron comentarios acerca de la dificultad, pero al tratarse de personas que no jugaban mucho, no los tuve en cuenta, ya que apenas tienen habilidad jugando.

Estas pruebas y correcciones han sido realizas en los siguientes dispositivos:

- Ordenador de sobremesa
  - I5-6600k 4.40GHz
  - 16 GB ddr4 ram
  - Grafica geForce GTX 970
- Portátil msi GP60 2PE
  - I5-4210H
  - 8GB ram
  - Grafica GT 840m
- Móviles
  - Samsung s7
  - Samsung s8
  - Tableta Huawei MediaPad M1

Tanto en el ordenador de sobremesa como en los Samsung s7 y s8 el juego ha ido fluido. Sin embargo, en el portátil (~3 años antigüedad) en muchas ocasiones el juego da tirones debido a que tiene bajadas de FPS. En la tableta Huawey el juego no va nada fluido en todo momento.

11.- Conclusiones

11.1.- Trabajos futuros

En este apartado hablaremos sobre posibles mejoras que se podrían añadir al juego. Algunas de ellas se plantearon en un principio, pero al final por falta de tiempo no han podido ser añadidas:

- Creación de nuevos niveles con diferentes objetivos y nuevos obstáculos
- Creación de trampas a lo largo del nivel, que afecten a cualquier personaje que las active
- Añadir más armas (bate de beisbol, francotirador, arma laser)
- Creación de un tutorial guiado. Es decir, que mientras juegues te salgan imágenes desplegables que te den consejos sobre que teclas pulsar
- Crear un menú que indique la vida que tienen los enemigos, como piensan, el daño que realizan etc.
- Crear un menú que explique en detalle el daño de cada arma, así como sus tiempos de reutilización.
- Crear la posibilidad de que al disparar hayan desmembramientos en los personajes enemigos.

11.2.- Conclusión del trabajo

La creación de este videojuego me ha enseñado como funciona realmente un videojuego. Durante el desarrollo me he dado cuenta de que muchas veces al jugar a otros juegos pensaba que las cosas se hacían de una forma y resulta que no, que no tiene nada que ver con lo que piensas.

Durante el desarrollo también he aprendido que crear un juego requiere más complejidad de lo que en un principio pueda parecer. En un principio no sé porque tenía la idea de que crear un juego era algo sencillo. Pero tras acabar este TFG, en mi opinión es algo más difícil y muchísimo más costoso que realizar cualquier cosa que tenga que ver con la plataforma web o al menos con lo que yo he realizado hasta el momento de dicha plataforma.

También me he dado cuenta de la gran importancia del equipo a la hora de elaborar juegos. Yo he tenido que dedicar bastantes horas de búsquedas para encontrar sonidos, texturas, crear efectos de partículas adecuados, buscar información etc. Y muchas veces esto me ha requerido mucho tiempo de aprendizaje. Así que tener a una persona que se dedique únicamente a esas áreas y encima sea un profesional es un punto muy a favor a la hora de desarrollar cualquier cosa.

Algo que me ha encantado durante el desarrollo ha sido la comunidad de Unity. Una comunidad enorme gracias a la cual he podido resolver todas mis dudas. Prácticamente en cualquier problema que me he metido ya le había aparecido anteriormente a otro

desarrollador, así que “simplemente” he tenido que aplicar las soluciones que ofrecían los usuarios.

Otra plataforma que me ha sorprendido ha sido youtube, sobre todo a la hora de crear efectos gráficos. Prácticamente todas las bases que he aprendido a la hora de crear efectos de partículas han sido gracias a esta plataforma.

Como aspecto negativo durante mi aprendizaje, lo único de lo que me podría quejar es que toda la información de calidad se encuentra en ingles y mi nivel no es que sea muy alto.

Tras realizar este proyecto, me ha entrado curiosidad por aprender a hacer juegos en HTML5 o alguno especifico de móviles, en especial realizar juegos que tengan que ver con el multijugador. Seguramente en verano me dedique a ello e intente hacer algún juego HTML5 online.

Algunos ejemplos de estos juegos:

https://www.truevalhalla.com/

12.- Bibliografía

12.1.- Documentación de Unity

- Manual de Unity
  - https://docs.unity3d.com/Manual/
- Video tutoriales de Unity
  - https://unity3d.com/es/learn/tutorials/
  - https://www.youtube.com/

12.2.- Recursos

- Sonidos
  - https://www.youtube.com/
- Texturas, materiales y animaciones
  - https://www.assetstore.unity3d.com
  - https://www.blendswap.com/
  - https://www.cgtrader.com
  - https://www.textures.com
  - https://www.turbosquid.com
  - http://www.cadnav.com
  - https://www.mixamo.com

12.3.- Simulación de multitudes

- http://www.memecs.org
- https://arongranberg.com/astar/features

12.4.- Información

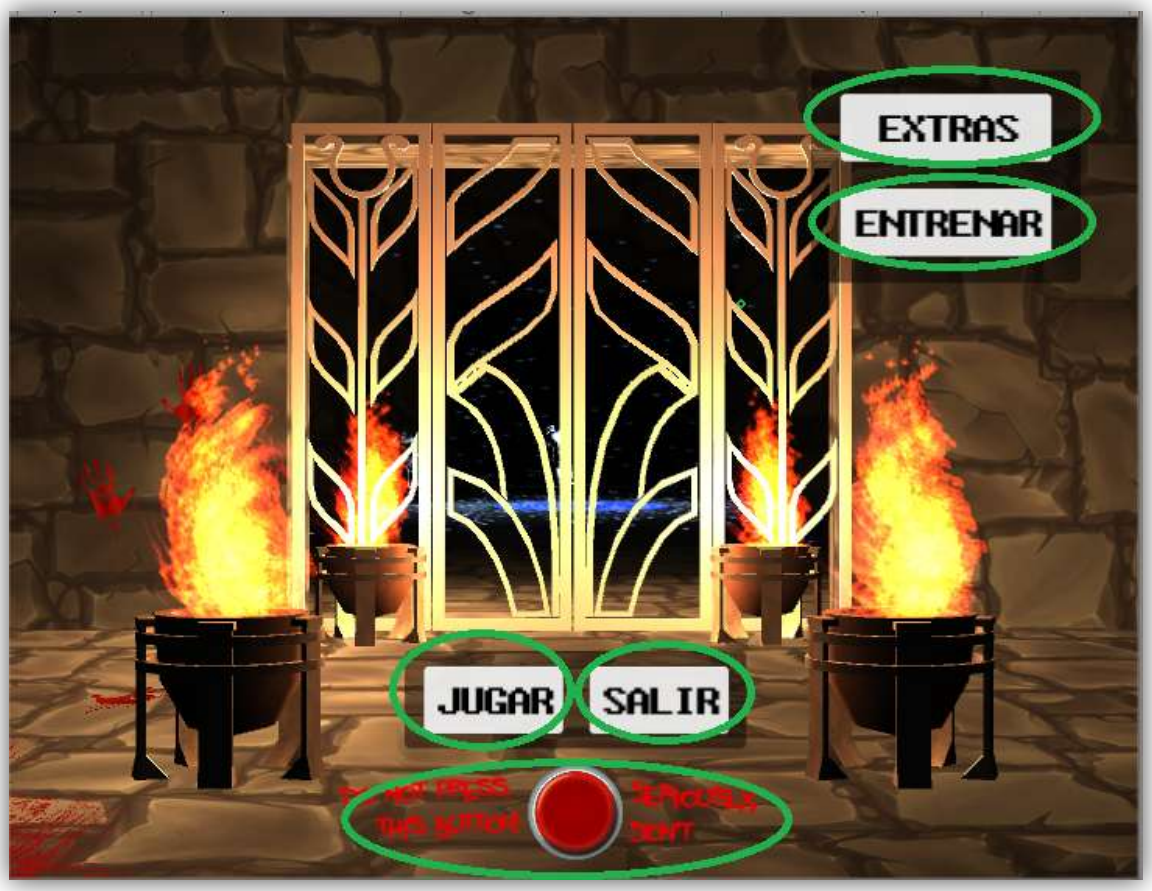
- https://es.wikipedia.org
- https://stackoverflow.com/
- https://histeriagamedev.wordpress.com/2017/06/11/consejos-y-buenas-practicas-en-unity/
- http://www.dev.org.es/publicaciones/libro-blanco-dev-2016
- Apuntes de la asignatura de fundamentos de los sistemas inteligentes para explicar el funcionamiento de los algoritmos de búsquedas de grafos (A*)

13.- Apéndices

13.1.- Manual de usuario

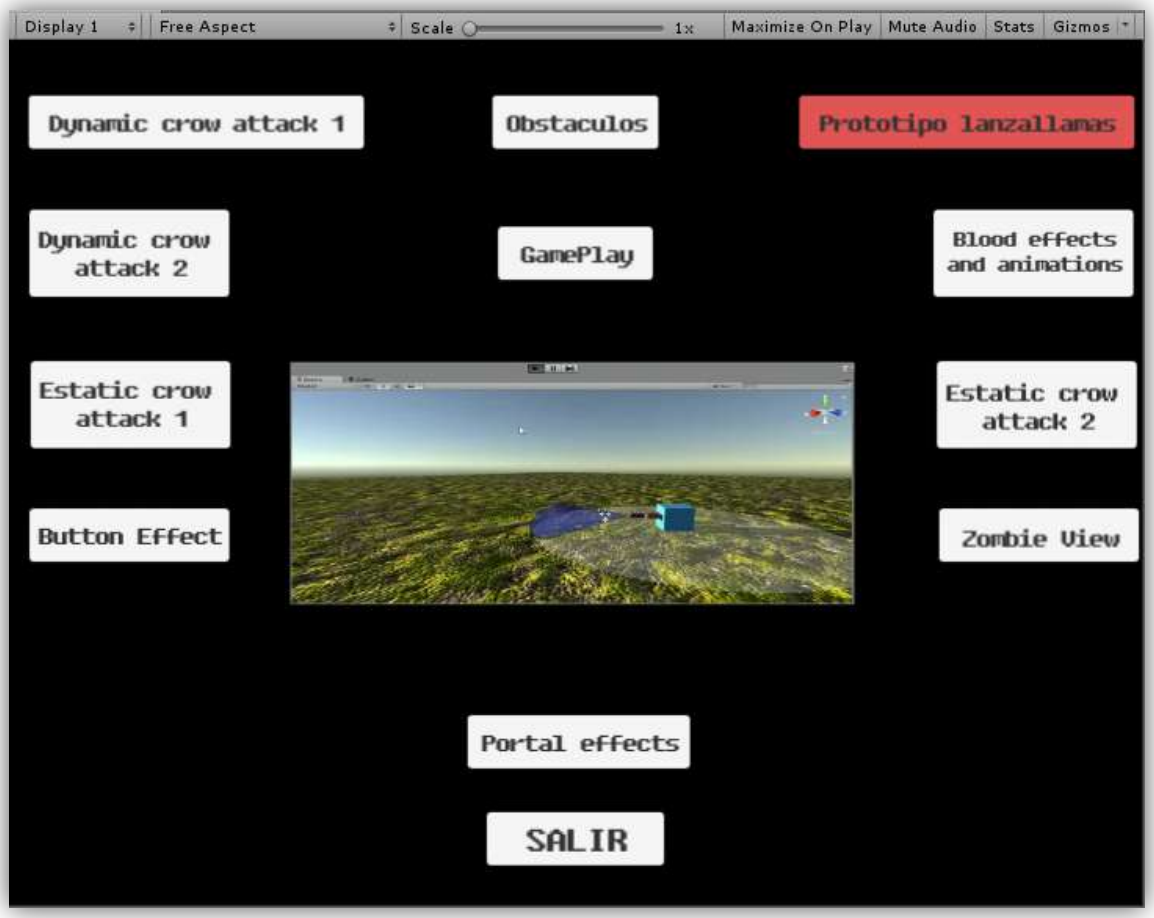
En el siguiente apéndice se añade un manual que enseñe a los jugadores de forma simple como jugar.

Pantalla de inicio



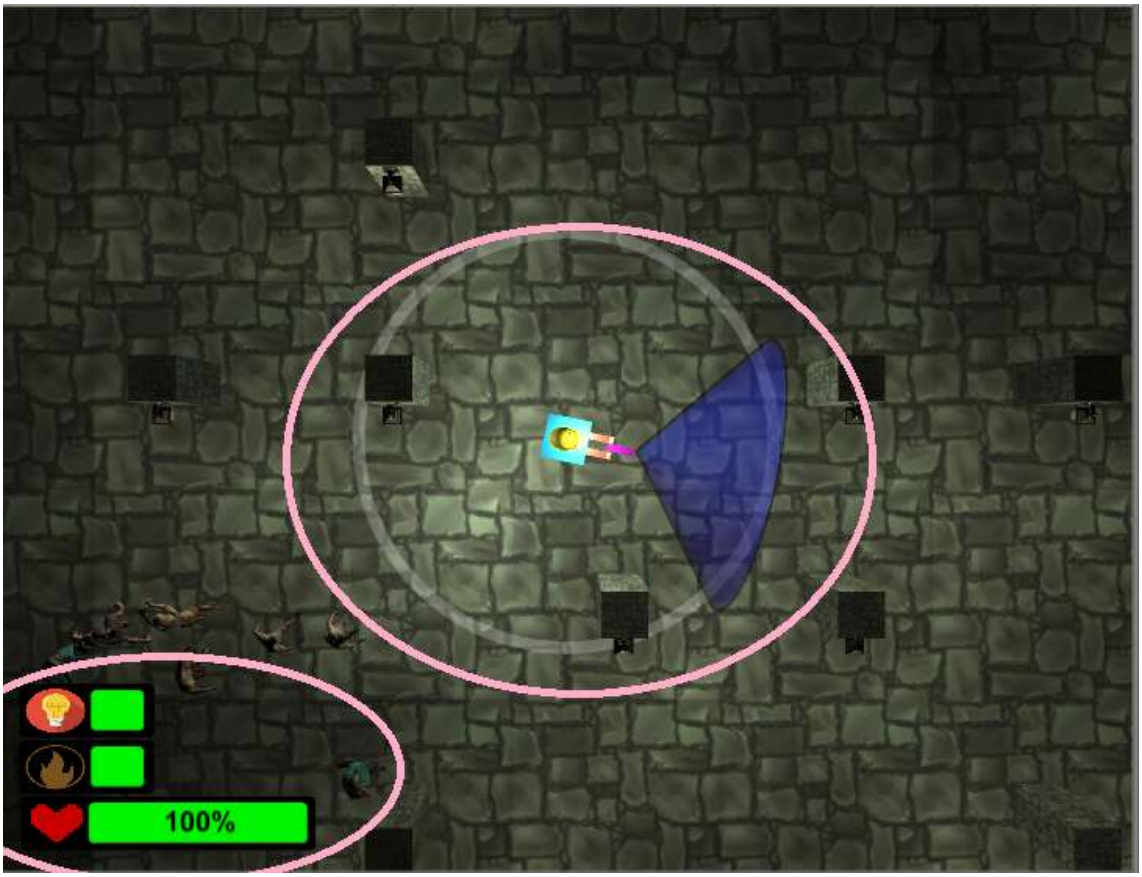
Opción	Descripción
Jugar	Inicia el primer y único
Salir	Cierra el juego
Botón rojo	Inicia un efecto de fuego que mata a los zombis que están tras la puerta
Extras	Muestra algunos de los videos creados durante el desarrollo del juego
Entrenar	Inicia el nivel de entrenamiento

Pantalla de extras



Cada botón tiene asociado un video con la temática del texto del botón. Basta con pulsar el botón y el video se reproducirá.

Nivel de entrenamiento



Opción	Descripción
Bombilla	Indica el estado actual de la luz del casco. El color verde indica que la has encendido. El color rojo indica que está apagada. Aunque la bombilla se encuentre en color verde, puede suceder que se funda la luz. Por lo que permanecerá X segundos apagada y luego se encenderá automáticamente.
Llama	Indica si el lanzallamas se puede usar o no. El color verde indica que está disponible. Rojo no está disponible.
Corazón	Indica la vida del jugador. Cuando llega a 0 el jugador muere y pierde la partida.
Circulo blanco semitransparente	Indica hasta donde llegan las llamas en un ángulo de 360º.
Área azul	Indica el área en la que se va a realizar daño.

Menú de pause

Opción	Descripción
Objetivos	Muestra los objetivos que se deben de cumplir para terminar el nivel
Controles	Muestra todos los controles disponibles
Apuntes	Indica pequeños apuntes sobre el juego
Salir	Permite volver a la pantalla de inicio
Pulsar “escape”	Cierra el menú de pause

Menú de controles de PC

Menú de controles de móvil

Menú de GameOver

Opción	Descripción
Reintentar	Reinicia el nivel
Salir	Permite volver a la pantalla de inicio