Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERIA INFORMÁTICA DE GESTIÓN Y SYSTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

2017 / 2018

Proyecto Labirint

Memoria

DATOS DE LA ALUMNA O DEL ALUMNO	**DATOS DEL DIRECTOR O DE LA DIRECTORA**
NOMBRE Bingen	NOMBRE Mikel
APELLIDOS Ros Garai	APELLIDOS Villamañe Gironés
	DEPARTAMENTO Lenguajes y sistemas informaticos
FDO.:	FDO.:
FECHA:	FECHA:

Proyecto Labirint

Índice

Índice Figuras

Índice de Tablas

1 - Introducción

Desde la creación del primer videojuego a manos de Notan Bushnell y Ted Dabney, el muy conocido Pong. La industria del videojuego ha ido creciendo sin parar año tras año, ha ido evolucionando desde las máquinas recreativas, las consolas de sobremesa hasta los smartphones. A pesar de ser una industria joven, ha pasado por varios períodos de auge y crisis. Hoy en día los videojuegos son más conocidos y usados que nunca, estos han pasado de ser un entretenimiento para unos pocos, a mover miles de personas e incluso crear profesiones nunca vistas hasta la fecha, entre ellos los jugadores profesionales o canales de televisión exclusivos para estos.

Origen del proyecto

Desde pequeño me han gustado los videojuegos y quería participar en la industria de los videojuegos, por lo que, cuando llegue al día de la jornada de puertas abiertas y se nos enseñó algunos proyectos entre los que se incluía un juego de naves hecho por unos alumnos quedé fascinado. Pensé que yo tambien queria saber hacer algo así y tomé la decisión de matricularme en esa carrera.
Durante los 4 años de carrera he ido gestando la idea del proyecto hasta lo que ha terminado por ser hoy en día.

Motivación

La primera motivación para la realización de este proyecto es poder usar los conocimientos obtenidos en la carrera y usar una herramienta desconocida y aprender a usarla de forma autodidacta.
La segunda motivación es poder crear un videojuego y que la persona que lo esté usando sea capaz de pasar un buen rato.
La tercera motivación es comprobar qué capacidades se pueden adquirir con el conocimiento obtenido durante los 4 cursos universitarios.

Contexto de negocio

Como se ha dicho previamente, la industria del videojuego está en alza y cada vez estos son más espectaculares, esto conlleva más trabajo y dinero. Por esta razón el trabajo de una sola persona no puede competir en la industria. Debido a esto, este proyecto no tiene ninguna posibilidad en el mercado, por ello, mi idea era usarlo en las jornadas de puertas abiertas de la universidad para enseñar a los alumnos las posibilidades que trae la carrera.
Será un juego hecho para plataformas windows y mac.

2 - Planteamiento inicial

Objetivos

Uno de los objetivos del proyecto, es entretener durante un periodo corto de tiempo al usuario, mediante un juego que a medida que se avanza en él se va haciendo más difícil y va poniendo a prueba las capacidades del usuario.
Otro objetivo principal es elegir una herramienta o tecnología nueva y aprender de forma autodidacta a usarla.
Por último, se tiene como objetivo hacer un juego en el que su dificultad vaya aumentando a medida que se juega y tenga varios elementos aleatorios para que el usuario no pueda predecir lo que va a pasar, a su vez, el jugador se enfrentará a diferentes grupos de enemigos mientras recorre las infinitas salas del juego.

Herramientas

Una de las herramientas principales del proyecto va a ser un equipo de sobremesa, Imprescindible para poder realizar el proyecto, tanto el código del juego como la documentación. Se va a usar un equipo de sobremesa porque es el ordenador más potente del que se dispone, pero podría usarse un portátil que soportase el software mencionado posteriormente.
Como sistema operativo se va a usar macOs sierra 10.12.6 ya que es el sistema que tiene instalado el equipo de sobremesa.
Como programa principal, se va a usar unity, que es un motor de videojuegos en el que se va a gestar todo el videojuego. Se va a usar la versión 5.5.1f1.

Existía otra posibilidad para usar como motor del juego, pero este aspecto se ve analizado en el capítulo-sección **4 - Elección de lenguajes y tecnologías**.

Para gestionar la documentación y el almacenamiento de los backups se va a usar google drive, pues es gratuito y su editor de texto es potente y fácil de usar. Se valoró usar dropbox, pero no dispone de un editor de documentos como el mencionado previamente. Para hacer los diagramas necesarios se va a usar el servicio web cacoo.

Juego

En este apartado se explicará de forma breve en qué consiste el juego:
El jugador tiene que avanzar por las diferentes salas del juego eliminando o esquivando enemigos. El mapa de cada sala se genera de forma aleatoria y tiene una distribución, número de enemigos y objetos aleatoria. Cuanto más progrese el jugador, más enemigos habrá y será más difícil avanzar hasta la siguiente sala mediante la señal de exit. Existen diferentes tipos de enemigos, y cada uno se comportará de forma distinta al encontrarse con el jugador.

EDT

Las siguientes ilustraciones representan el esquema de descomposición del proyecto en diferentes tareas y una breve descripción de estas.

Esquema del proyecto

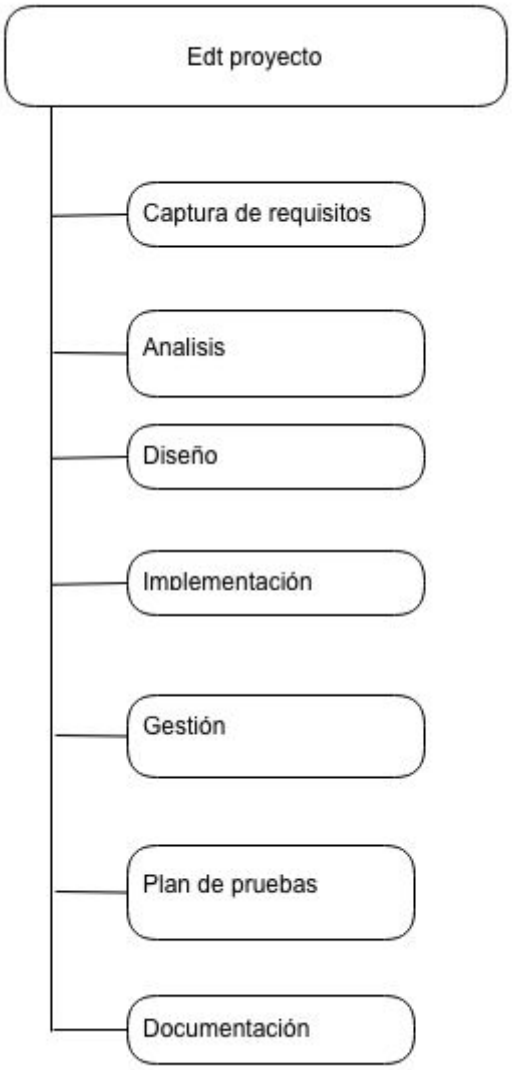

Ilustración 1 - Esquema proyecto EDT

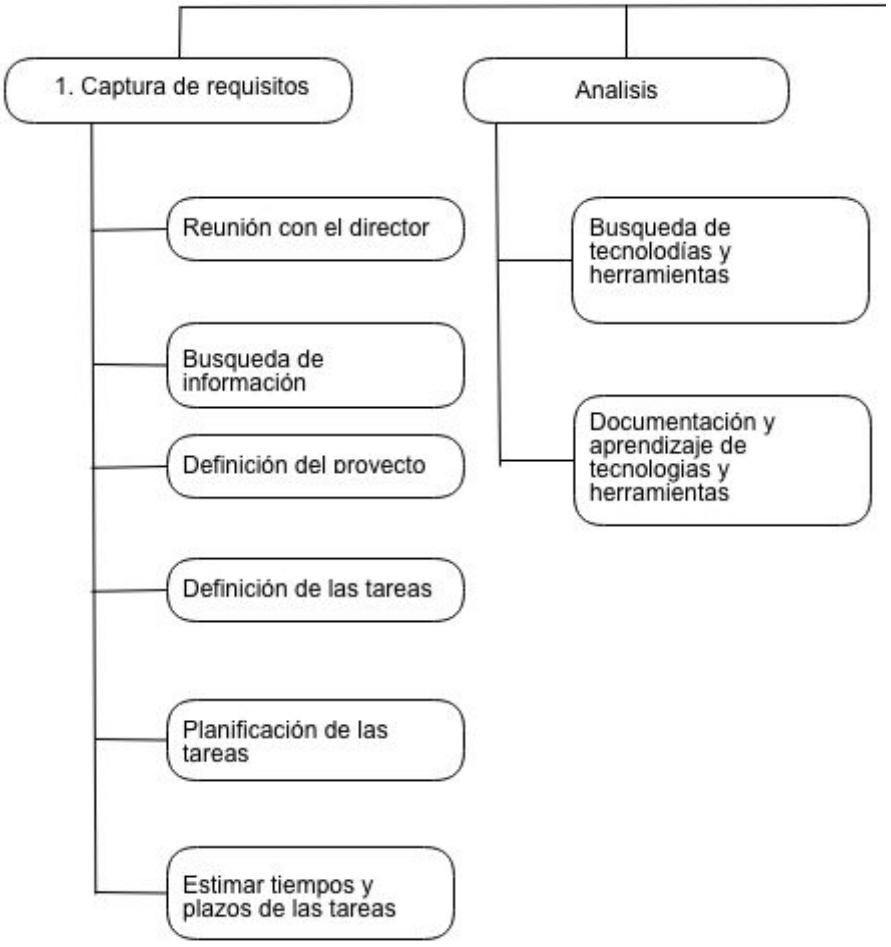

Ilustración 2 - Parte 1 de 3 EDT

Captura de requisitos

Reunión con el director

- **Descripción:** Primera toma de contacto, para hablar sobre el proyecto.

Búsqueda información

- **Descripción:** Búsqueda de información sobre las herramientas y tecnologías que se van a necesitar y elegir una entre las alternativas posibles.

Definición del proyecto

- **Descripción:** Se define la estructura de la aplicación en las herramientas y tecnologías seleccionadas.

Definición de las tareas

- **Descripción:** Definir las tareas necesarias para la realización del proyecto.

Planificación de las tareas

- **Descripción:** Planificar el orden de las tareas definidas.

Estimar tiempos y plazos de realización de las tareas

- **Descripción:** Realizar una estimación de los tiempos necesarios para cada tarea.

Analisis

Búsqueda de tecnologías y herramientas

- **Descripción:** Realizar una búsqueda entre las distintas tecnologías y herramientas que puedan ser de utilidad en el proyecto.

Documentación y aprendizaje de tecnologías

- **Descripción:** Aprender a utilizar y familiarizarse con las herramientas y tecnologías escogidas.

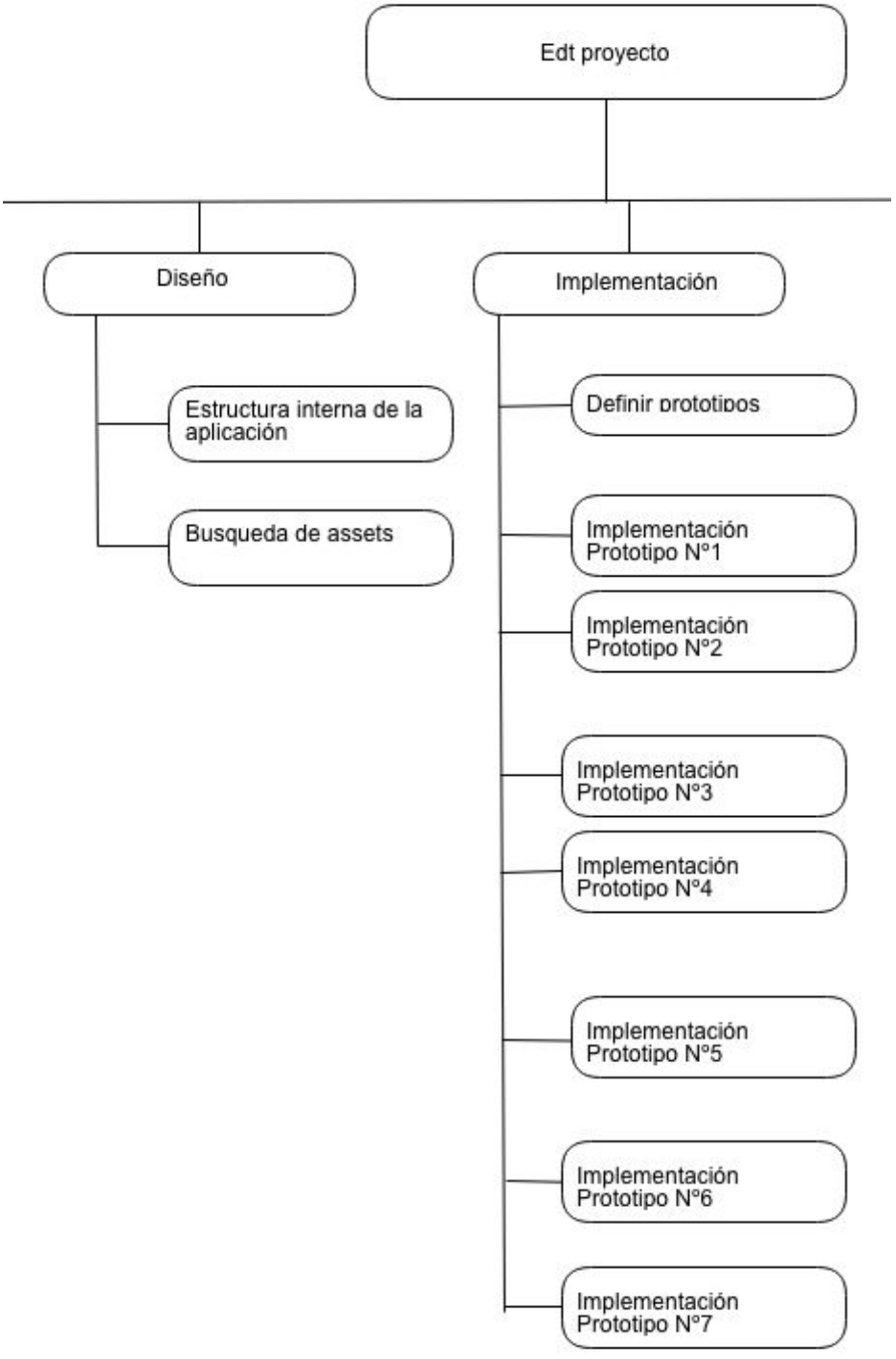

Ilustración 3 - Parte 2 de 3 EDT

Diseño

Estructura interna de la aplicación

- **Descripción:** Determinar una estructura eficiente, teniendo en cuenta las necesidades del proyecto.

Busqueda de assets

- **Descripción:** Buscar y descargar sprites con licencia de reutilización para poder usar de forma gratuita.

Implementación

Definir prototipos

- **Descripción:** Definir las partes en las que se va a segmentar el proyecto para facilitar su implementación.

Prototipo Nº1 “crear e inicializar Prefabs”

- **Descripción:** Se creará y ajustará complementos y animaciones de los enemigos, jugador, objetos del suelo y las paredes.

Prototipo Nº2 “crear gestor y mapa del juego”

- **Descripción:** Se creará el gestor del juego y mapa donde transcurre el juego.

Prototipo Nº3 “movimiento”

- **Descripción:** Se implementará la funcionalidad de movimiento para los enemigos y el personaje controlable.

Prototipo Nº4 “colisiones”

- **Descripción:** Se implementarán las funcionalidades de las colisiones entre los objetos del juego y sus respectivas interacciones.

Prototipo Nº5 “menús”

- **Descripción:** Se implementarán los menús del juego.

Prototipo Nº6 “añadir componente de aleatoriedad”

- **Descripción:** Añadir aleatoriedad en los mapas, crear mapas de tamaños dinámicos aleatorios y número aleatorio de salas por cada nivel.

Prototipo Nº7 “añadir más enemigos”

- **Descripción:** Añadir nuevos enemigos con rutinas especiales.

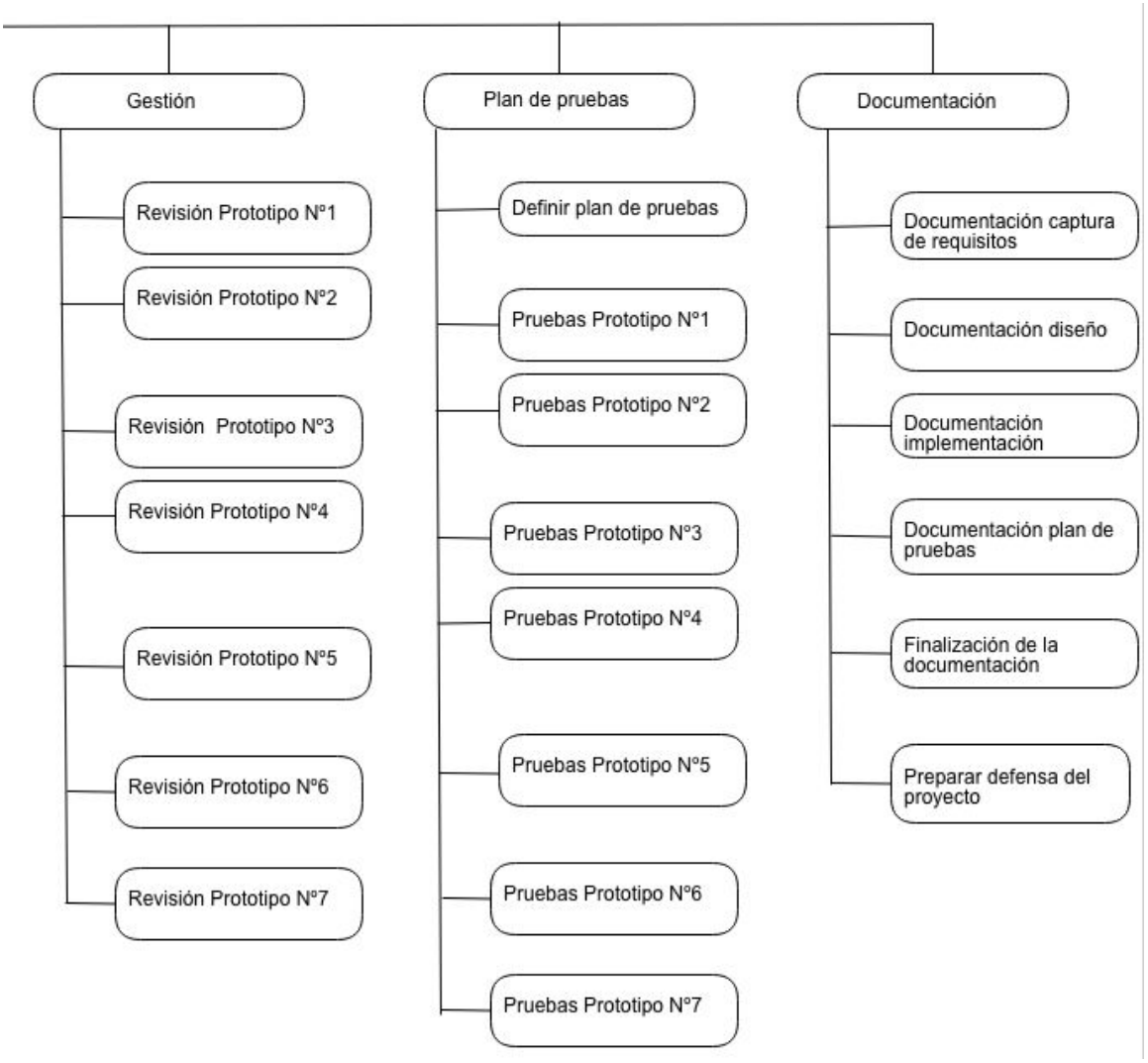

Ilustración 4 - Parte 3 de 3 EDT

Gestión

Revisión prototipo Nº1

- **Descripción:** Revisión de la implementación del prototipo.

Revisión prototipo Nº2

- **Descripción:** Revisión de la implementación del prototipo.

Revisión prototipo Nº3

- **Descripción:** Revisión de la implementación del prototipo.

Revisión prototipo Nº4

- **Descripción:** Revisión de la implementación del prototipo.

Revisión prototipo Nº5

- **Descripción:** Revisión de la implementación del prototipo.

Revisión prototipo Nº6

- **Descripción:** Revisión de la implementación del prototipo.

Revisión prototipo Nº7

- **Descripción:** Revisión de la implementación del prototipo.

Plan de pruebas

Definir plan de pruebas

- **Descripción:** Definir un plan de pruebas para los diferentes prototipos.

Pruebas prototipo Nº1

- **Descripción:** Se ejecutará un plan de pruebas para comprobar el correcto funcionamiento del prototipo.

Pruebas prototipo Nº2

- **Descripción:** Se ejecutará un plan de pruebas para comprobar el correcto funcionamiento del prototipo.

Pruebas prototipo Nº3

- **Descripción:** Se ejecutará un plan de pruebas para comprobar el correcto funcionamiento del prototipo.

Pruebas prototipo Nº4

- **Descripción:** Se ejecutará un plan de pruebas para comprobar el correcto funcionamiento del prototipo.

Pruebas prototipo Nº5

- **Descripción:** Se ejecutará un plan de pruebas para comprobar el correcto funcionamiento del prototipo.

Pruebas prototipo Nº6

- **Descripción:** Se ejecutará un plan de pruebas para comprobar el correcto funcionamiento del prototipo.

Pruebas prototipo Nº7

- **Descripción:** Se ejecutará un plan de pruebas para comprobar el correcto funcionamiento del prototipo.

Documentación

Documentación captura de requisitos

- **Descripción:** Realizar la documentación de la sección de captura de requisitos.

Documentación del diseño

- **Descripción:** Realizar la documentación de la sección de análisis y diseño.

Documentación de la implementación

- **Descripción:** Realizar la documentación de la sección de la implementación

Documentacion del plan de pruebas

- **Descripción:** Realizar la documentación de la sección del plan de pruebas.

Finalización de la documentación

- **Descripción:** Completar la memoria del proyecto añadiendo la información recopilada durante el desarrollo de este.

Preparar defensa

- **Descripción:** Preparar la defensa del proyecto.

En el siguiente apartado se muestra la estimación de tiempo que se le ha hecho a cada tarea.

Captura de requisitos

Reunión con el director

- **Duración:** 1 hora

Búsqueda información

- **Duración:** 8 horas

Definición del proyecto

- **Duración:** 4H

Definición de las tareas

- **Duración:** 5 horas

Planificación de las tareas

- **Duración:** 4 horas

Estimar tiempos y plazos de realización de las tareas

- **Duración:** 3 horas

Análisis

Búsqueda de tecnologías y herramientas

- **Duración:** 2 horas

Documentación y aprendizaje de tecnologías

- **Duración:** 20 horas

Diseño

Estructura interna de la aplicación

- **Duración:** 15 horas

Busqueda de assets

- **Duración:** 2 horas

Implementación

Definir prototipos

- **Duración:** 5 horas

Prototipo Nº1 “crear e inicializar Prefabs”

- **Duración:** 25 horas

Prototipo Nº2 “crear gestor y mapa del juego”

- **Duración:** 25 horas.

Prototipo Nº3 “movimiento”

- **Duración:** 35 horas

Prototipo Nº4 “colisiones”

- **Duración:** 20 horas

Prototipo Nº5 “menús”

- **Duración:** 2 horas

Prototipo Nº6 “añadir componente de aleatoriedad”

- **Duración:** 30 horas

Prototipo Nº7 “añadir más enemigos”

- **Duración:** 30 horas

Gestión

Revisión prototipo Nº1

- **Duración:** 1 horas

Revisión prototipo Nº2

- **Duración:** 1 horas

Revisión prototipo Nº3

- **Duración:** 1 horas

Revisión prototipo Nº4

- **Duración:** 1 horas

Revisión prototipo Nº5

- **Duración:** 1 horas

Revisión prototipo Nº6

- **Duración:** 1 horas

Revisión prototipo Nº7

- **Duración:** 1 horas

Plan de pruebas

Definir plan de pruebas

- **Duración:** 3 horas

Pruebas prototipo Nº1

- **Duración:** 2 horas

Pruebas prototipo Nº2

- **Duración:** 3 horas

Pruebas prototipo Nº3

- **Duración:** 3 horas

Pruebas prototipo Nº4

- **Duración:** 3 horas

Pruebas prototipo Nº5

- **Duración:** 1 horas

Pruebas prototipo Nº6

- **Duración:** 5 horas

Pruebas prototipo Nº7

- **Duración:** 3 horas

Documentación

Documentación captura de requisitos

- **Duración:** 15 horas

Documentación del diseño

- **Duración:** 5 horas

Documentación de la implementación

- **Duración:** 10 horas

Documentación del plan de pruebas

- **Duración:** 6 horas

Finalización de la documentación

- **Duración:** 20 horas

Preparar defensa

- **Duración:** 20 horas

En la siguiente tabla se muestra el cálculo estimado de horas necesarias para realizar todas las tareas para el cumplimiento de las tareas mencionadas en el apartado anterior.

Tareas	Estimación (Horas)
Captura de requisitos	25
Reunión con el director	1
Búsqueda de información	8
Definición del proyecto	4
Definición de las tareas	5
Planificación de las tareas	4
Estimar tiempos y plazos	3
Analisis	22
Busqueda de tecnologías y herramientas	2
Documentación y aprendizaje de las tecnologias	20
Diseño	17
Estructura interna de la aplicación	15
Búsqueda de assets	2
Implementación	150
Definir prototipos	5
Implementación prototipo Nº1	20
Implementación prototipo Nº2	20
Implementación prototipo Nº3	25
Implementación prototipo Nº4	15
Implementación prototipo Nº5	5
Implementación prototipo Nº6	30
Implementación prototipo Nº7	30
Gestión	7
Revisión prototipo Nº1	1
Revisión prototipo Nº2	1
Revisión prototipo Nº3	1
Revisión prototipo Nº4	1
Revisión prototipo Nº5	1
Revisión prototipo Nº6	1
Revisión prototipo Nº7	1

Plan de pruebas	23
Definir plan de pruebas	3
Pruebas prototipo Nº1	2
Pruebas prototipo Nº2	3
Pruebas prototipo Nº3	3
Pruebas prototipo Nº4	3
Pruebas prototipo Nº5	1
Pruebas prototipo Nº6	5
Pruebas prototipo Nº7	3
Documentación	76
Captura de requisitos	15
Diseño y analisis	5
Implementación	10
Plan de pruebas	6
Finalización de la codumentación	20
Defensa del proyecto	20
Total	320

Tabla 1 - Estimación tiempo de tareas

La estimación indica que se tardarán 320 horas en finalizar el desarrollo del proyecto. Teniendo en cuenta que la media de horas de trabajo a la semana es de 15 horas, la duración del proyecto será de 21 semanas

	Junio	Julio				Agosto
Tareas	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6
Captura de requisitos						
Reunión con el director						
Busqueda de información						
Definición del proyecto						
Definición de las tareas						
Estimar tiempos y plazos						
Analisis						
Busqueda de tecnologias y herramientas						
Documentación y aprendizaje de las tecnologias						
Implementación						
Definir prototipos						
Implementación prototipo Nº1						
Implementación prototipo Nº2						
Implementación prototipo Nº3						
Implementación prototipo Nº4						
Implementación prototipo Nº5						
Implementación prototipo Nº6						
Implementación prototipo Nº7						
Gestión						
Revisión prototipo Nº1						
Revisión prototipo Nº2						
Revisión prototipo Nº3						
Revisión prototipo Nº4						
Revisión prototipo Nº5						
Revisión prototipo Nº6						
Revisión prototipo Nº7						
Plan de pruebas						
Definir plan de pruebas						
Pruebas prototipo Nº1						
Pruebas prototipo Nº2						

Ilustración 5 Parte 1 de 4 GANTT

					Septiembre
Tareas	Semana 7	Semana 8	Semana 9	Semana 10	Semana 11
Implementación prototipo Nº3					
Implementación prototipo Nº4					
Implementación prototipo Nº5					
Implementación prototipo Nº6					
Implementación prototipo Nº7					
Gestión					
Revisión prototipo Nº3					
Revisión prototipo Nº4					
Revisión prototipo Nº5					
Revisión prototipo Nº6					
Revisión prototipo Nº7					
Plan de pruebas					
Pruebas prototipo Nº3					
Pruebas prototipo Nº4					
Pruebas prototipo Nº5					
Pruebas prototipo Nº6					

Ilustración 6 - Parte 2 de 4 GANTT

Tareas	Semana 12	Semana 13	Semana 14	Semana 15	Semana 16
Implementación prototipo Nº6					
Implementación prototipo Nº7					
Gestión					
Revisión prototipo Nº6					
Revisión prototipo Nº7					
Plan de pruebas					
Pruebas prototipo Nº6					
Pruebas prototipo Nº7					

Ilustración 7 - Parte 3 de 4 GANTT

Tareas	Semana 17	Semana 18	Semana 19	Semana 20	Semana 21
Documentación					
Captura de requisitos					
Diseño y análisis					
Implementación					
Plan de pruebas					
Finalización de documentación					
Defensa del proyecto.					

Ilustración 8 - Parte 4 de 4 GANTT

Gestión de riesgos

En el siguiente apartado, se analizan los posibles riesgos que pueden ocurrir durante el proyecto, como intentar preverlos y en caso de que ocurran que hacer al respecto.

- Problemas con el software de unity

Probabilidad: Baja-Media
Impacto: Extremo
Consecuencia : Retraso en la ejecución de las tareas.
Prevención: Usar la última versión estable del software unity
Plan de contingencia: Migrar el proyecto a la siguiente versión estable.

-Pérdida de datos guardados

Probabilidad: Media-Alta
Impacto : Extremo
Consecuencia : Retraso en la ejecución de las tareas.
Prevención: Realización de backups cada 5 días
Plan de contingencia: Restaurar el proyecto desde el último backup

-Pérdida de datos sin guardar

Probabilidad: Alta
Impacto : Bajo
Consecuencia : Retraso en la ejecución de las tareas.
Prevención: Guardar el proyecto cada 30-45 mins.
Plan de contingencia: Rehacer el trabajo perdido.

-Problemas con el hardware

Probabilidad: Baja
Impacto : Alta
Consecuencia : Retraso en la ejecución de las tareas.
Prevención: Dar buen uso al hardware.
Plan de contingencia: Cambiar de hardware lo antes posible.

-Problemas de implementación

Probabilidad: Alta
Impacto : Media
Consecuencia : Retraso en la ejecución de las tareas.
Prevención: Dedicar tiempo al aprendizaje de la herramienta.
Plan de contingencia: Usar los foros de unity.

Viabilidad económica

En el siguiente apartado se muestra el análisis de la viabilidad económica del proyecto.

Personal

El proyecto va a ser creado por una persona cualificada, asumiendo que el sueldo por hora de un programador es de 22€/hora y asumiendo que el proyecto está estimado a realizarse en 320 horas. La cantidad es de 7040€.

Hardware

-Ordenador sobremesa

Se usará un equipo de sobremesa valorado en 1700€ con una vida media de 10 años y un uso del 5%

Amortización anual 1700€/10 años = 170€/Año

Gasto del ordenador ((170x21) * 5%) / 52 semanas = 8,5€

Software:

Licencia Unity : Gratuita
Licencia OS X : Incluida en los gastos del equipo de sobremesa.

Gastos totales	Euros
Personal	7040
Hardware	8,5
Equipo de sobremesa	8,5
Software	0
Total	**7048,5**

Tabla 2 - Viabilidad económica

El proyecto no está destinado a un uso comercial, por lo que no se contempla la posibilidad de ningún beneficio económico.

3 - Antecedentes

Durante los más de 40 años en los que han existido los videojuegos, han ido creándose una gran cantidad de generos y subgeneros para estos., entre todos estos, este proyecto pertenece al subgénero de los “Roguelike”.
Las características más habituales en este tipo de juegos son las siguientes:

- Juegos para un solo jugador
- Énfasis en el contenido aleatorio(Mapas, enemigos, objetos…)
- La jugabilidad es el aspecto primario por encima de los gráficos o la accesibilidad.
- Muerte permanente, es decir, no existen puntos de guardado. En caso de perder la partida se vuelve a empezar desde el principio.
- Dificultad elevada y aumenta con cada nivel.
- Un juego sencillo en lo que se refiere a la narrativa.
- El objetivo es llegar lo más lejos posible ,avanzar lo máximo posible en la mazmorra.

El primero de estos juegos, fue Rogue(1980), dando este nombre a sus sucesores, ”roguelike” significa parecido a rogue.
Este proyecto se ha inspirado casi por completo en 1 solo *roguelike*, Nuclear throne, que contiene todas las características principales del subgénero y añadiendo una estética post apocalíptica muy marcada.

Ilustración 8 - Nuclear Throne(2015)

El objetivo de nuclear throne es avanzar el máximo número de niveles posible para poder llegar al jefe final y elegir la opción de derrotarlo y terminar la partida o hacer loop.
Hacer loop consiste en cumplir ciertos requisitos antes de derrotar al jefe final, una vez derrotado volvemos al piso 1 de la mazmorra con todo el equipo previamente obtenido para volver intentar llegar al final. Este proyecto no contiene jefe final, por lo que no hay una opción de terminar la partida sin morir o reiniciar.

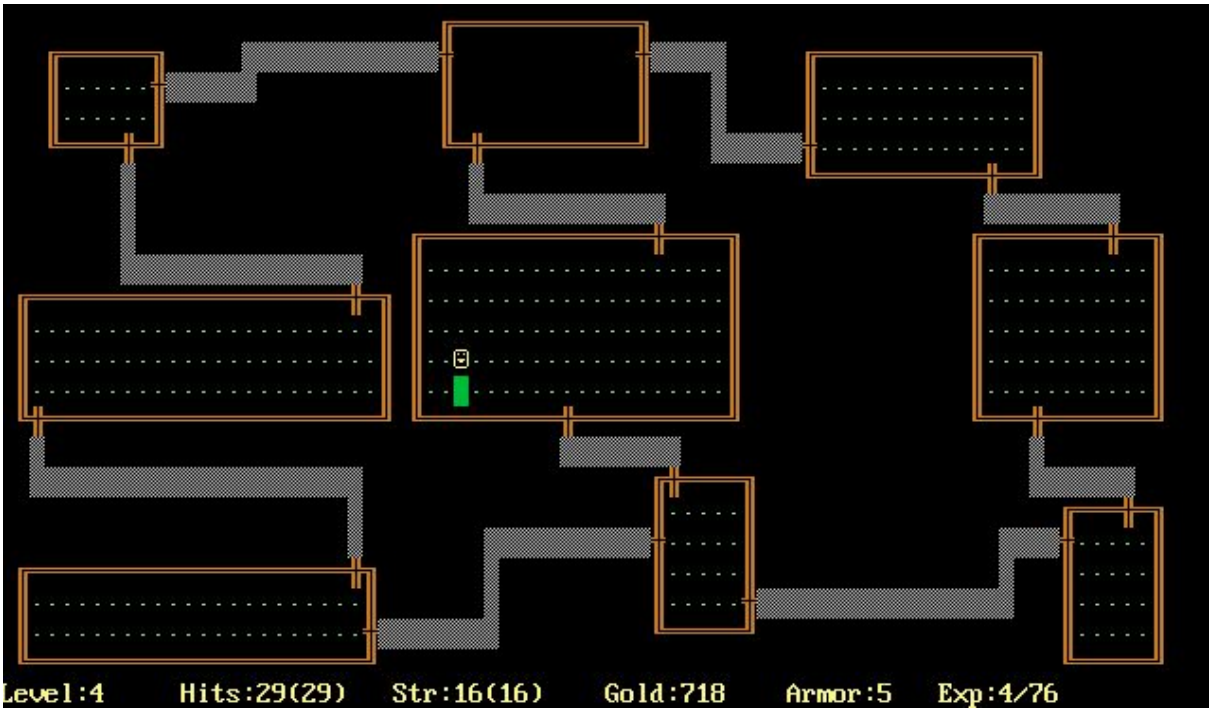

Ilustración 9 - Rogue(1980)

4 - Elección de lenguajes y tecnologías

A la hora de elegir que programa y que lenguaje de programación usar, se han tenido en cuenta varias opciones, en el siguiente sección se explicará qué decisiones se han tomado y el porqué de estas.

Lo primero que se tuvo que tener en cuenta es el tipo de juego que se iba a hacer.
Entre los muchos generos y subgeneros de los videojuegos, se ha decidido hacer uno que pertenece al subgénero roguelike.
Es un juego que se puede hacer lo básico de forma fácil, y complicar el desarrollo tanto como se quiera añadiendo ideas nuevas al juego.
Una vez elegido el ámbito del juego, hay que elegir si será en 2D o en 3D.
Los expertos recomiendan siempre empezar con un juego en 2D ya que empezar programando en 3 dimensiones es más complicado y puede acabar frustrando.
Además de esto, los juegos roguelike tienden a ser 2D. Por esto se eligió el 2D.

Una vez decidido el género del juego, era necesario elegir unas herramientas adecuadas para el desarrollo de este. A la hora de elegir el motor del juego se han tenido en cuenta 2 opciones: Unity y Unreal engine. Estos dos, son los programas para creación de videojuegos más famosos. Pero la razón por la que se han tenido en cuenta, es por qué son gratuitos y muy potentes.

Ilustración 10 - Ejemplo juego creado por unity (***Chaos Reborn*** 2015)

Ilustración 11 - Ejemplo juego creado por unreal engine (***Rock of Ages*** 2011)

A la hora de elegir motor de juego, lo primero que hay que tener en cuenta es el tipo de juego que se va a hacer a hacer. Como el proyecto es un juego en 2D es recomendado usar Unity ya que está más especializado en estos. A su vez, los expertos recomiendan empezar en el mundo del videojuego con Unity.
Por lo que se decidió usar Unity en vez de Unreal Engine.

Unity ofrece la posibilidad de programar en C# o en Python, por lo que hace falta elegir cual de los 2 se adapta más al proyecto.

Ambos son lenguajes orientados a objetos, pero las cualidades en las que sobresalen son distintas. C#, o C Sharp, es un lenguaje de programación excelente para crear aplicaciones GUI (Graphical user interface), mientras que python destaca más en el ámbito científico y de computación.

Además de esto, uno de los objetivos del proyecto es aplicar los conocimientos adquiridos en la carrera y aprender a usar una herramienta desde 0.

Teniendo en cuenta todo lo mencionado antes, C# es el lenguaje de programación que se ha elegido para el desarrollo del proyecto.

5 - Captura de requisitos

En este apartado se mostrarán y explicarán la jerarquía de actores, casos de uso y modelo de dominio.

Jerarquía de actores

Player

Es el actor que se encarga de ejecutar las acciones que tienen que ver con el personaje controlado por el jugador.

Casos de uso

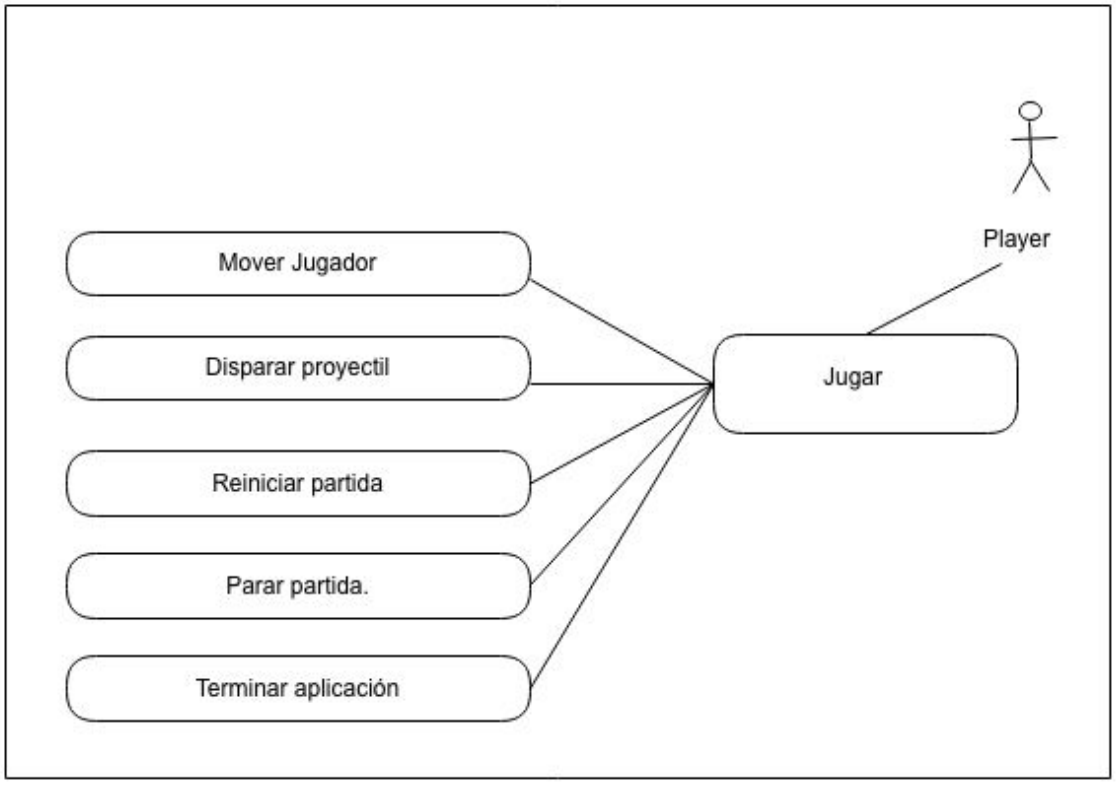

Ilustración 12 - Casos de uso Player

Mover jugador

Recurso disponible para el actor Player que permite moverse por la pantalla al jugador pulsando las flechas del teclado o las teclas ‘a’, ‘s’, ‘d’ y ‘w’.

Disparar proyectil

Recurso disponible para el actor Player que permite crear un proyectil al hacer click con el ratón y disparar en dirección a la posición actual del ratón.

Reiniciar partida

Recurso disponible para el actor Player que permite reiniciar la partida.

Parar partida

Recurso disponible para el actor Player que mediante el menú de pausa, detener el tiempo de juego.

Terminar aplicación

Recurso disponible para el actor Player que permite terminar la aplicación.

Diagrama de Clase 1.0

Al principio del proyecto se hizo un diagrama de clase para estructurar la aplicación y su desarrollo.

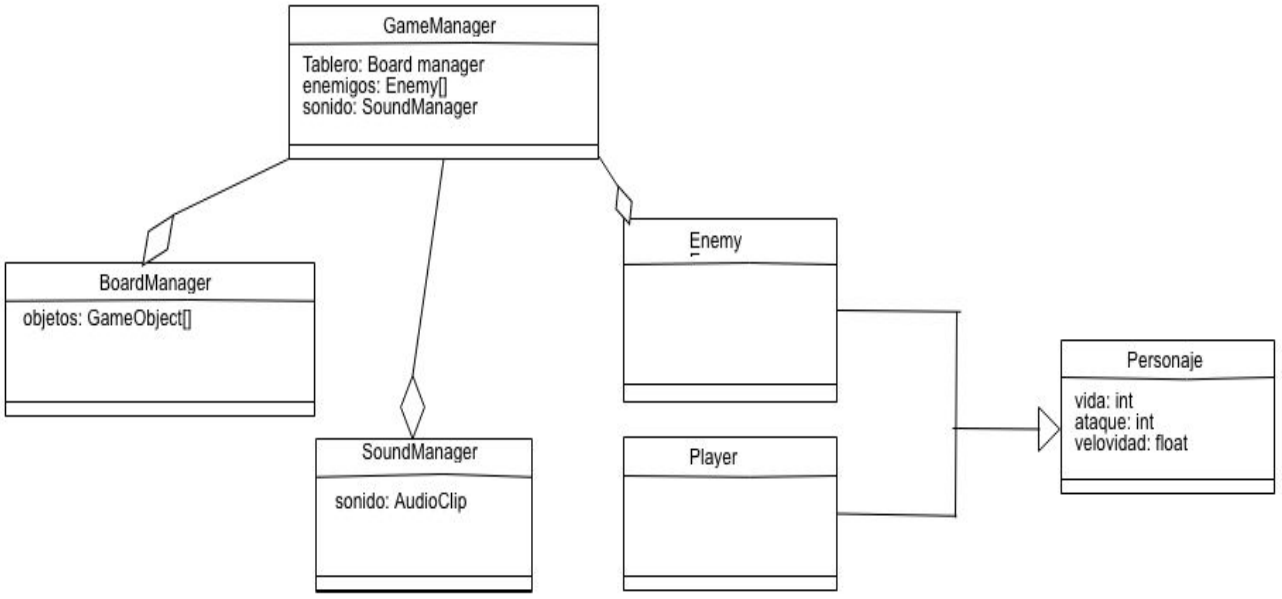

Ilustración 13 - Diagrama de clase 1.0

Player

Player representa al personaje controlable por el jugador, y gestiona sus acciones e interacciones con el mapa y enemigos.

Enemy

Enemy representa la IA de los enemigos a los que se enfrenta el jugador, este script gestiona sus acciones e interacciones con el mapa y el jugador

GameManager

GameManager representa al administrador del tablero de juego, gestiona los diferentes ámbitos del juego para que sea posible jugar.

BoardManager

BoardManager representa el gestor del tablero de juego, este, crea y organiza el mapa del nivel y sus contenidos.

SoundManager

SoundManager representa el gestor de la música y sonidos del juego.

Personaje

Personaje, representa los metodos y parametros en común que tienen todos los personajes del juego,el movimiento por ejemplo. Todos los gameobject de la escena que pertenecen a la layer units serán denominados personajes.

6 - Análisis y Diseño

En el siguiente apartado se explica la estructura interna que ha seguido y los componentes usados en el proyecto con la herramienta unity 2D.

Escena o *scene*

Es el elemento donde se juntan todos los elementos para poder crear la escena del juego.

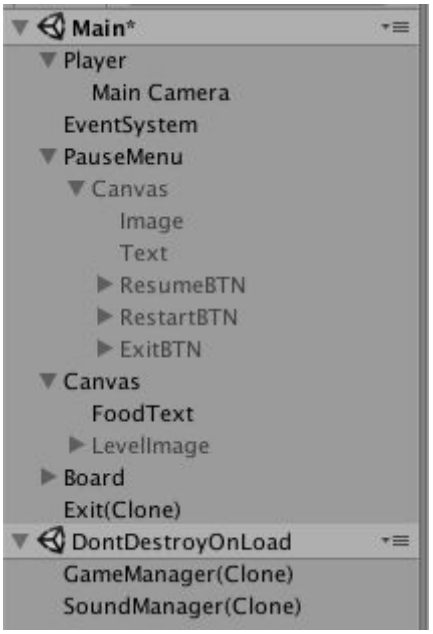

Ilustración 14 - Escena

Gameobject

Es uno de los conceptos más básicos e importantes de unity, cada objeto del juego es un gameobject, pero de por si solos no hacen nada, hay que configurarlos y darle los parámetros necesarios para que se conviertan en lo que sea necesario.
Dependiendo del tipo de objeto que se quiera crear se le añadirán una clase de componentes u otros.

Componentes usados para los Gameobject:

Rigidbody2D

Es un componente usado por el motor de físicas del juego, es similar al Rigidbody, pero este al ser 2D solo usa los planos XY.
Rigidbody2D gestiona la posición, rotación y la posibilidad de crear colisiones con otros objetos, entre otras cosas.

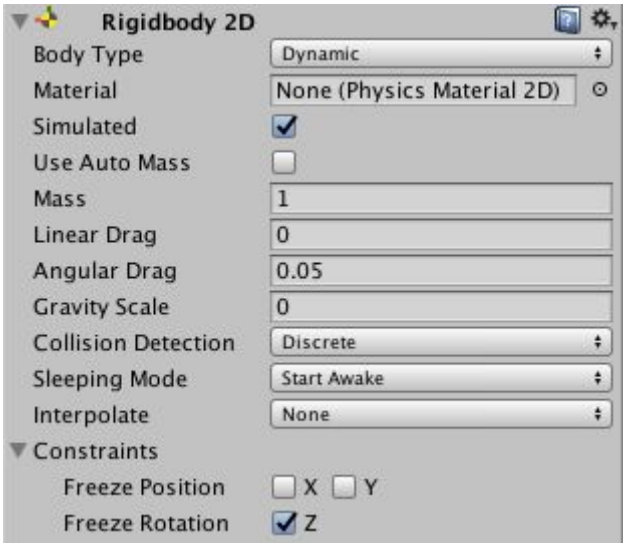

Ilustración 15 - RigidBody2D

Collider2D(Box & circle)

Son las figuras geométricas que limitan las colisiones

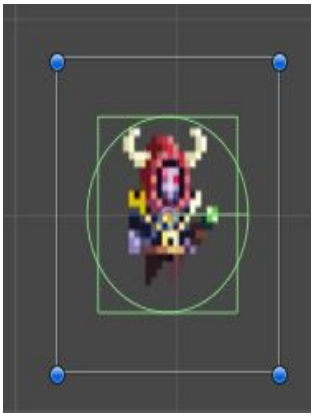

Ilustración 16 - Box y Circle Collider

Tag

Es una etiqueta, un string, que se asigna al gameobject para poder identificarlo.

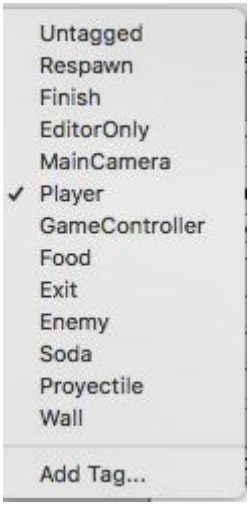

Ilustración 17 - Tag display

Layer

Los distintos layers del juego son las distintas capas que tiene, se usa para colocar los elementos de forma que se puedan ver todos.

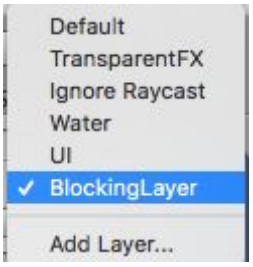

Ilustración 18 - Layer display

Animation

Al juntar un grupo de imágenes estas crean una animación

Imágenes:

Ilustración 19 - Sprites enemigo

Animator Controller

Al añadir animaciones a un gameobject, de forma automática se crea un animator controller para gestionarlas.
Con este podemos crear transiciones entre ellas, usando triggers para pasar de una a otra.

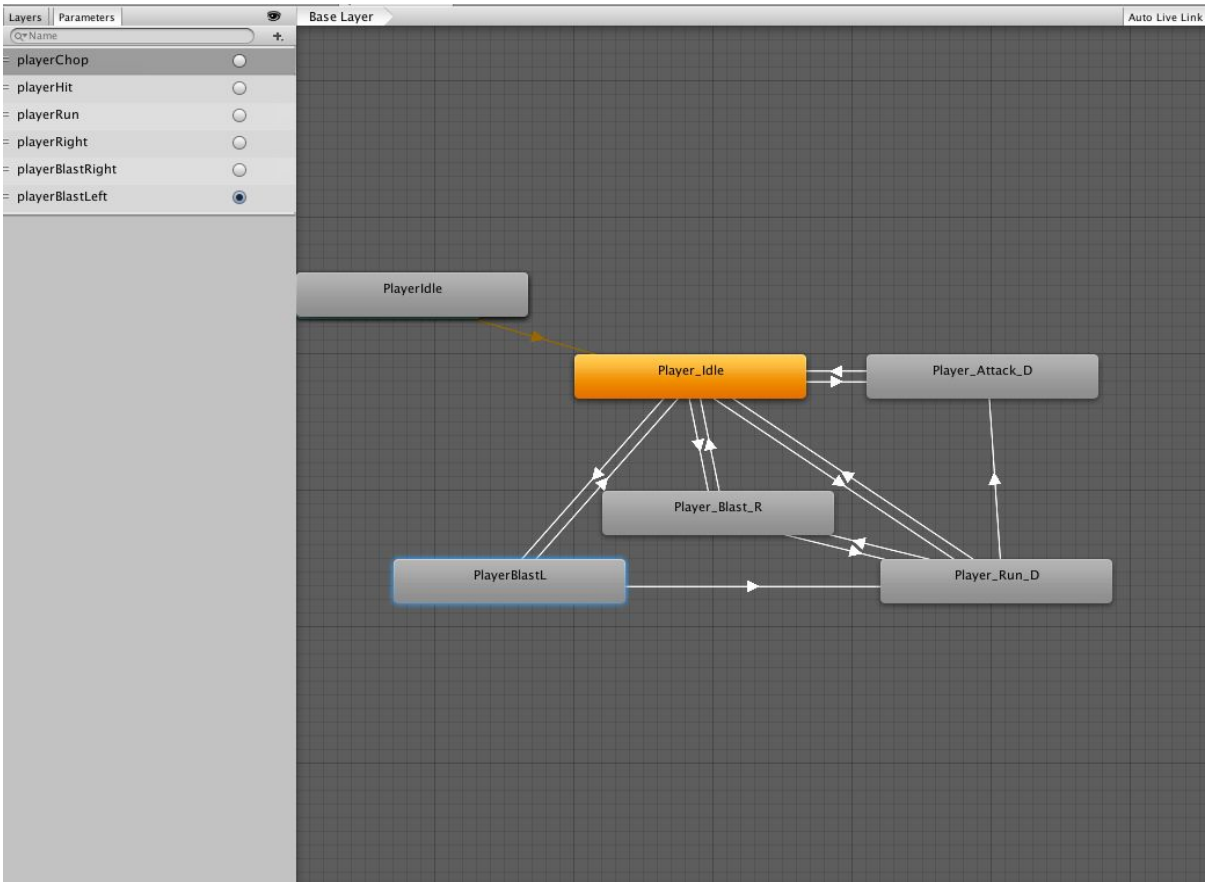

Ilustración 20 - Animator controller

Prefab

Los prefabs son un instrumento de unity para guardar gameobjects previamente configurados y poder usarlos en la escena, o *scene*

Scritps

Son los guiones desde los que se gestionan los gameobject.
Para este proyecto se han escrito en C#.

Camera

Es el dispositivo que captura las imágenes del juego para mostrarla al jugador.

Canvas

Es un gameobject específico en el que se guardan los elementos de la *UI (User interface) como los menús del juego.*

Diagrama de clase 2.0

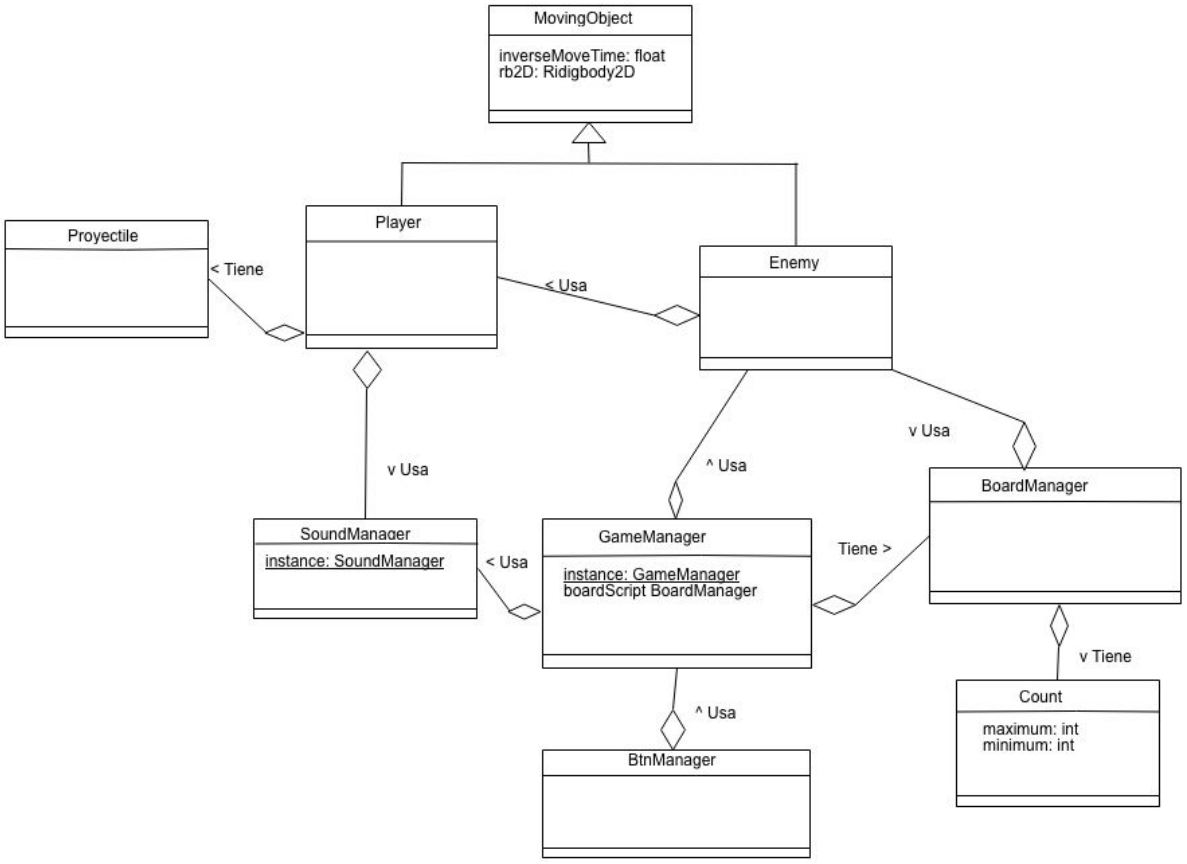

Ilustración 21 - Diagrama de clase 2.0

MovingObject

Se encarga de gestionar los métodos de movimiento que son comunes entre todos los personajes. Como la forma de moverse de todos los personajes del juego son iguales, todos heredan esta clase para no tener que repetir el mismo método en más de 1 clase.

Player

Se encarga de gestionar e inicializar al personaje controlado por el jugador. Gestiona cuando tiene que moverse,inicializa el cuando disparar y como reaccionar al colisionar con diferentes objetos.

Proyectile

Se encarga de gestionar las colisiones de los proyectiles creados por el personaje controlado por el jugador.

Enemy

Se encarga de gestionar la IA de los enemigos, cada enemigo tiene un string que marca su tipo y dependiendo de este sus rutinas de movimiento y formas de ataque serán distintas. Además de esto, gestiona las diferentes reacciones que tienen los enemigos al chocar contra diferentes objetos.

GameManager

Es un singleton que se encarga de gestionar el juego. Desde ordenar la creación del mapa hasta cuando se tienen que mover los enemigos.

BoardManager

Se encarga de gestionar el mapa en el que se juega. Inicializa el mapa desde el suelo y paredes hasta los enemigos o la señal de salida.

BtnManager

Se encarga de gestionar el menú de pausa y sus diferentes acciones, al que se accede pulsando “esc”.

SoundManager

Es un singleton que se encarga de gestionar los sonidos y la música del juego.

7 - Desarrollo

En este capítulo se explicará las diversas fases que ha habido durante el desarrollo y las decisiones que se han tomado a la hora de elegir distintos métodos y complementos.

En la creación del proyecto, se pueden diferenciar 2 partes del desarrollo.
La primera es desde unity, en la que creamos y configuramos los objetos y complementos que vamos a usar.

Ilustración 22 - Imagen unity

La segunda es la edición de los scripts usados por los objetos. Para esto, se ha usado un programa que viene instalado con unity, monodevelop

Ilustración 23 - Imagen monodevelop

Las dos fases del desarrollo ocurren de forma combinada, es decir, no se hace primero todo en unity y al terminar se pasa a monodevelop. Se va alternando entre los dos programas.

Ejemplo: Primero, se crea un gameobject del proyectil con un script y sus demás complementos luego se edita en monovedelop.

Lo primero del desarrollo ha sido crear el gameobject relacionado con el personaje controlable por el jugador.
Para la creación de cualquier gameobject, lo inicial es crear un empty gameobject.
GameObject -> create empty

Una vez creado, hay que renombrarlo y añadirle los complementos que se crean necesarios, en este caso un RigidBody2D y un BoxCollider2D.
Add component -> RigidBody2D, Add component -> BoxCollider2D

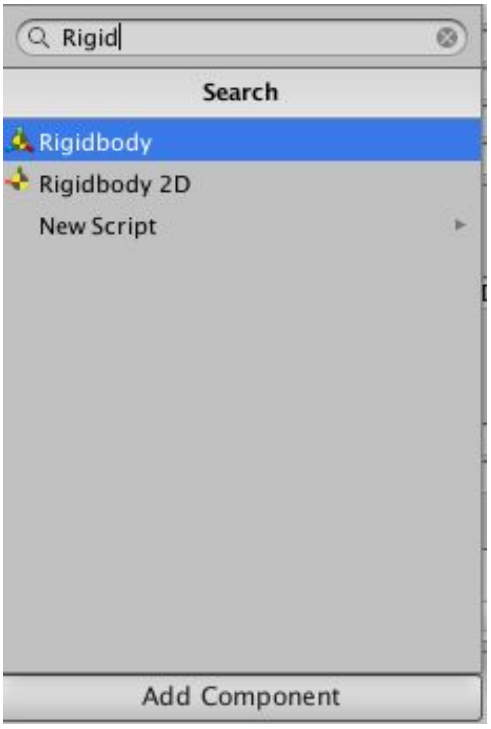

Ilustración 24 - Add Component

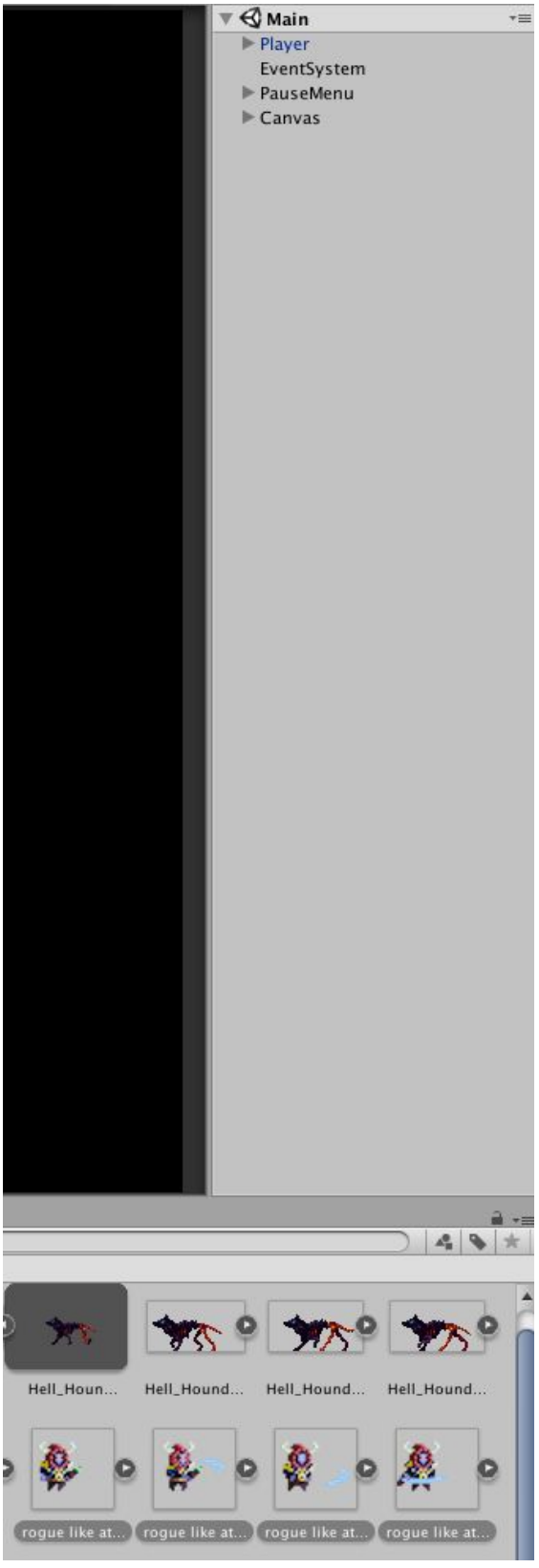

Ilustración 25 - Crear animaciones

El siguiente paso es añadirle las animaciones que tengamos disponibles.

(Seleccionar las imágenes que forman la animación y arrastrarlas al gameobject).

Al crear la primera animación se crea un Animator Controller, que sirve de gestor de estas, permitiendo crear transiciones entre ellas y ajustar parámetros tales como su duración o si pueden ser interrumpidas por otras acciones.
Por último seleccionar el tag del objeto en caso de que lo tenga y su layer.
tag=player & layer=blockinglayer.

Para la creación de los objetos que serán los enemigos, el proceso es similar al anterior, únicamente cambiando las animaciones y el nombre del gameobject.
tag=enemy & layer=blockinglayer

El proceso de creación del gameobject de paredes y objetos es el mismo que el anterior, pero estos son objetos estáticos por lo que no necesitan un RigidBody2D ni animaciones, en su lugar tendrán una sola imagen y un BoxCollider2D.

(***Soda, Food & Exit Signal)***
Tag=(food || soda || wall || untagged || exit)
layer= (default || wall)

El gestor del juego, BoardManager es un script de C# (***Create -> C# script***). Una vez creado no se altera desde unity. Se usa monodevelop.

En cuanto al GameManager, se crea el gameobject pero como complementos solo se le añaden scripts.
Component -> script

Para crear el gestor de sonido del juego se crea el gameobject y se le añaden los complementos Audio Source (2x).

Al igual que el el player, para crear los proyectiles, se crea el gameobject y se le añaden BoxCollider2D y RigidBody2D, ya que este es un objeto que va a colisionar con otros y va a moverse por el mapa, a su vez se le añade una animación.

Para gestionar el movimiento se usa el script MovingObject, como se ha mencionado antes, para editar los scripts se usa el programa monodelevop.
Player y Enemy heredan de este los métodos para poder moverse por el mapa.

Para poder mover los gameobject se usan los métodos que proporciona el complemento Rigidbody2D. Para ello es necesario un Vector2 que señale la dirección y un float que indique la velocidad de movimiento.

Para gestionar las diferentes colisiones es necesario tener, como mínimo, un Collider2D en el objeto que tenga el script.

Usando los métodos ***Ontrigger(Collider)*** conseguimos el objeto con el que se ha chocado.
Podemos sacar su tag para saber con qué tipo de objeto se ha hecho la colisión y actuar en consecuencia.

Para que el juego tenga sentido, los enemigos deben actuar de forma inteligente para ponerle las cosas difíciles al jugador.
Cada gameobject de enemigo tiene su propio e irrepetible tag con el que se diferencia de los otros. Dependiendo de este tag sus ataques o patrones de movimiento serán distintos.
El Gamemanager gestiona cuando los enemigos deben moverse, y los llama mediante el método ***MoveEnemy()***. Cada enemigo obtendrá la posición del Player y tendrá que juzgar si está a su alcance y si va a atacarle huir o acercarse.

Existen 2 grupos básicos de enemigos, los que atacan a distancia (shooter) o los que atacan cuerpo a cuerpo (melée). Dependiendo del tipo de enemigo este atacara de una u otra forma.

Desde el BoardManager se guardan todos los objetos posibles, comestibles, paredes suelo y enemigos. Este crea un mapa de dimensiones aleatoria.
Lo primero coloca el suelo en cada casilla y en los bordes del mapa coloca las paredes, Al terminar de crear el suelo y paredes, coloca de forma aleatoria un número aleatorio de objetos y enemigos de forma aleatoria. Por último coloca la señal de exit para poder avanzar al siguiente nivel.

Como todos los juegos de su género, la aleatoriedad es una pieza esencial.
Por ello el BoardManager crea una sala de un tamaño aleatorio en su número de columnas y filas. A su vez crea un cantidad aleatoria de salas por mapa.

A continuación, se muestran dos ejemplos del mapa al que tiene que enfrentarse el jugador en el nivel 5.

Ilustración 26 - mapa nivel 5 ejemplo 1

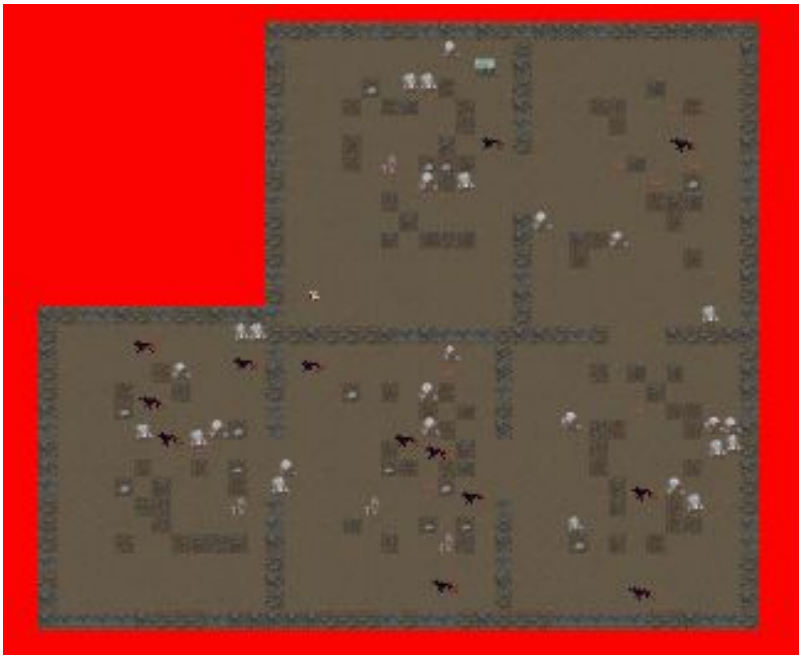

Ilustración 27 - mapa nivel 5 ejemplo 2

Este apartado tiene como finalidad explicar el porqué se han tomado ciertas decisiones a la hora de implementar o elegir diferentes parámetros. El funcionamiento de las distintas opciones es similar, pero en cada caso, una se ajusta más a lo que es necesario o consume menos recursos. Por lo que elegir bien en cada caso es esencial.

Inicializar un gameobject

A la hora de inicializar un gameobject, hay disponibles 3 opciones:
Start(), ***Awake()*** o no inicializarlos.

Se usará la 3. opción, no inicializarlos, cuando no se requiera ningún método o parámetro inicial para el correcto funcionamiento del gameobject o no tenga un gameobject que iniciar.
Por ejemplo BoardManager o BtnManager.

El método awake() se ejecuta antes de que empiece el juego, por ello, es necesario que se ejecute en las clases que sean necesarias para crear el juego. GameManager y SoundManager.
Por lo tanto, el método start(), se usa con los objetos que no son prioritarios para la creación del juego.

Actualizar objeto

A la hora de actualizar un objeto existen 2 posibilidades ***Update()*** o ***FixedUpdate()***.
El método ***Update()*** es llamado una vez por frame(Imagen por segundo) mientras que ***FixedUpdate()*** es llamado de forma fija cada cierto tiempo.
Se ha usado ***FixedUpdate()*** en los objetos que requieren del motor de físicas, ya sea para el movimiento o para otra acción.
De haber usado ***Update()*** en su lugar, en caso de que hubiese una bajada de frames en la pantalla por cualquier motivo, el tiempo de respuesta por parte del ***Update()*** sería menor que ***FixedUpdate()***. Por ello solo se ha usado ***Update()*** en los objetos en los que no depende del motor de físicas: *BtnManager*.
Mientras que ***FixedUpdate()*** en lo que sí lo necesita *Player* y *GameManager*.

El objeto Enemy no tiene método para actualizarse, porque se encarga GameManager de gestionar sus actualizaciones.

Collider2D

Existen varias opciones en cuanto a Colliders, pero las que se han tenido en cuenta han sido BoxCollider2D y CircleCollider2D.
Ha habido casos en los que ha habido necesidad de usar dos colliders. En esos casos se han usado un BoxCollider2D y un CircleCollider2D ya que no se puede tener dos Collider2D iguales.
En los casos en los que solo era necesario uno, se ha utilizado el que resultaba más óptimo para el rendimiento, que según se ha demostrado es CircleCollider2D.

Coger teclado

A la hora de recoger el input del teclado para crear los proyectiles se han barajado 2 posibilidades, usar el método ***GetButton(string)*** o ***GetButtonDown(string)***.
La diferencia entre estos dos métodos, es que ***GetButtonDown(string)*** recoge cuando se ha pulsado la tecla en cuestión, mientras que ***GetButton(string)*** es llamado mientras se mantenga la tecla pulsada.
En el caso de la creación de disparos, no se ha decidido si disparar el proyectil al hacer click con el ratón o mientras se mantenga el click sigan saliendo proyectiles.

Gestión de colisiones

Se han usado 2 métodos para gestionar las colisiones ***OnTriggerEnter2D(Collider)*** y ***OnTriggerStay2D(Collider)***. El primero de estos, se ejecuta cuando dos objetos chocan. Este se ha usado para los proyectiles, ya que una vez que han hecho contacto, deben hacer su función y luego destruirse.

Por otro lado, la segunda opción se activa y se mantiene activo mientras los 2 objetos estén en colisión. Para las colisiones entre el jugador y los enemigos se ha usado este método.

El Player tiene colisiones contra objetos que desaparecen al tocarlos, por ello lo recomendable seria usar ***OnTriggerEnter2D(Collider)***, pero como no se puede tener 2 metodos ***Ontrigger...(Collider)*** a la vez y para las colisiones contra los enemigos necesita el Stay es necesario gestionar todas sus colisiones con este.

Tipo de cuerpo para el objeto (RigidBody2D)

Hay 2 posibilidades a elegir, Dinamic o kinematic(Dinámico o cinemático).
Un objeto Dinámico, es afectado por las fuerzas tales como la masa o la gravedad(Simuladas). Mientras que los objetos cinemáticos no se ven afectado por estas. Además de esto, un RigidBody2D cinemático no acciona los métodos que gestionan las colisiones, por lo que atraviesa los objetos. Por ello, se han usado RigidBody2D dinámicos.

8 - Verificación y evaluación

En el siguiente apartado, se detallara como se ha procedido para comprobar el correcto funcionamiento del juego y una evaluación con un grupo pequeño de usuarios.

Unity facilita hacer las pruebas de las variables mediante las variables globales.
Es posible una vez compilado el juego, cambiar el valor de las variables públicas sin necesidad de recompilar todo, por lo que se ahorra mucho tiempo.
Se han llevado a cabo las pruebas relacionadas con variables que afecten a los prefabs principalmente, como el número de columnas en el mapa, velocidad de los enemigos o el daño que hacen por ataque.

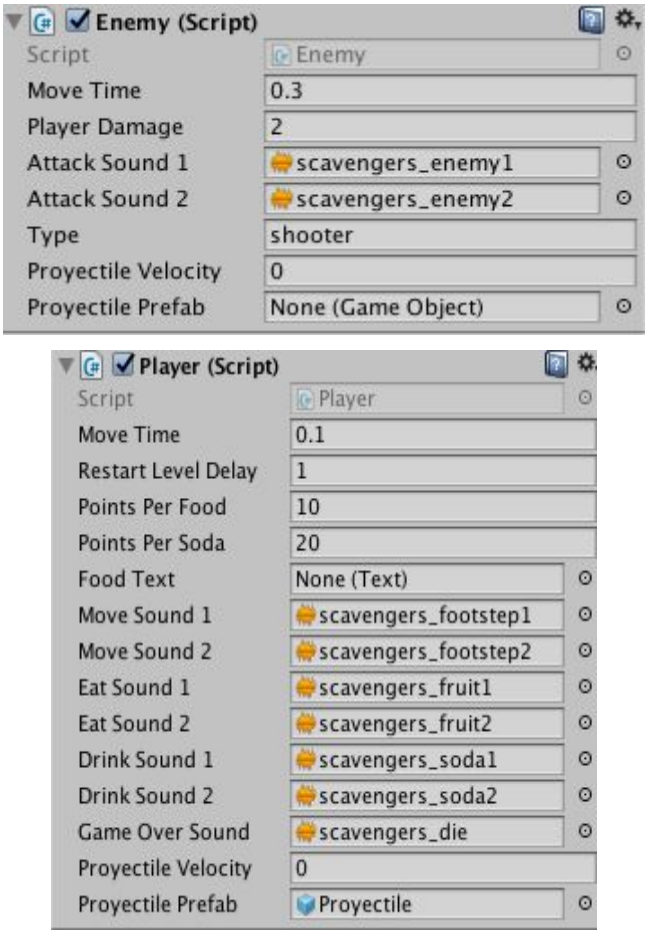

Ilustración 28 - Variables globales de un enemigo y el jugador

Por ejemplo: Para configurar la velocidad de movimiento del jugador, se ha usado la variable global Move Time dentro del prefab de Player, así, se puede probar sin tener que cambiarlo directamente en el código.

Debug

Las pruebas que no se pueden hacer mediante las variables globales se han hecho usando la herramienta de debug que trae unity.
Mediante el método ***Debug.Log(string)***, podemos pasar a la consola de debug el mensaje que necesitemos.

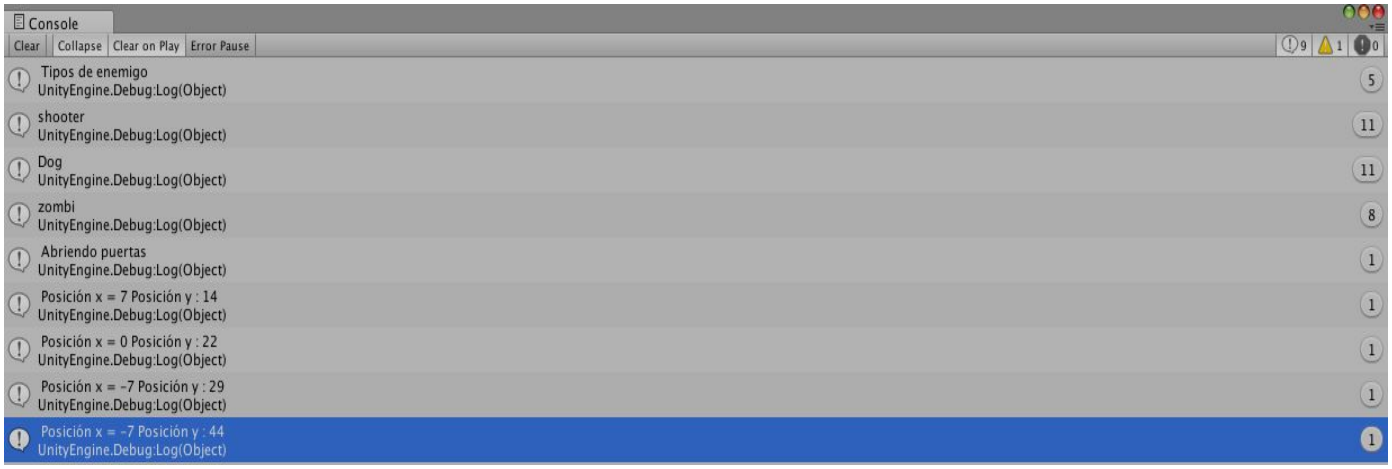

Ilustración 29 - Consola debug

En la captura, se puede ver los tipos de enemigo en el mapa, y la posición de las puertas entre las distintas salas.

El número al lado, es el cantidad de veces que se ha ejecutado el debug, es decir, hay 11 enemigos *shooter* 11 *Dog* y 8 *zombi*.

Durante la implementación de cada prototipo, para comprobar su correcto funcionamiento se han realizado diferentes tipos de pruebas.

En el primer prototipo, crear e inicializar prefabs, se realizaron las pruebas ejecutando la aplicación. De esta forma, se puede apreciar de forma visual si el resultado es el esperado o hace falta algún cambio.

El segundo prototipo, crear gestor y mapa del juego, tuvo unas pruebas similares a las del primer prototipo, es decir, se ejecutaba la aplicación para comprobar si el resultado era el esperado. En este caso, se usaban scripts de C# por ello, para ahorrar tiempo, las pruebas de las variables se hicieron con las variables globales. Pudiendo cambiar estas desde unity sin tener que recompilar los scripts de C#.

En el tercer prototipo, movimiento, el objetivo era crear un movimiento fluido para las unidades, desde el personaje controlado por el jugador hasta los enemigos.
Al igual que las pruebas de los prototipos anteriores, gran parte de las pruebas se hicieron mediante la aplicación para ver si el movimiento se ajustaba a lo que se buscaba. Por otra parte, para comprobar si las variables de movimiento eran adecuadas se ha usado la consola de debug.

El cuarto prototipo, colisiones, siguió la estructura de pruebas del tercer prototipo, aplicación y debug. Se comprobaba que las unidades realizasen las acciones requeridas al entrar en colisión con diferentes objetos o al mantenerse en colisión con estos.

Las pruebas del quinto prototipo, menús, se hizo de forma visual, iniciando la aplicación y probando que cada botón cumplia de forma correcta su función asignada. Sin embargo, el último de los botones, cerrar aplicación, no se puede probar desde unity por lo que fue necesario crear un ejecutable de la aplicación para poder probarlo.

Para poder realizar las pruebas del sexto prototipo, hubo que usar todas las herramientas disponibles, principalmente la consola de debug para comprobar que las variables de la aleatoriedad eran correctas y no se salían de los límites establecidos.
A su vez, se usó mucho la aplicación para comprobar que los elementos que se añadían, como distintas salas y puertas entre ellas, estaban colocados de forma correcta.

Por último, el prototipo 7, más enemigos, las pruebas se han realizado mediante la aplicación, es decir, jugando al juego.

Formulario

Para medir el grado de agrado del juego se ha creado el siguiente formulario

Te ha entretenido (Del 1 al 10, siendo 1 nada y 10 lo máximo)

1	2	3	4	5	6	7	8	9	

Lo recomendarías

1	2	3	4	5	6	7	8	9	

¿Qué es lo que menos te ha gustado?

-

¿Qué es lo que más te ha gustado?

-

¿Qué añadirías?

-

¿Has visto algún error o bug?

-

¿Qué mejorarías?

-

En el siguiente apartado, se analiza los resultados del formulario realizado a distintos grupos de usuarios. No se ha tenido en cuenta, género del usuario o conocimientos informáticos ya que no resultan relevantes para los objetivos del proyecto.
Las pruebas se han hecho con 15 usuarios, de entre 21-29 años y se han buscado usuarios que regularmente, mínimo una vez a la semana, jueguen a videojuegos.

En el gráfico mostrado a continuación se muestra el grado de entretenimiento que ha causado el juego en los usuarios. Siendo el 1 la peor nota posible y el 10 la mejor. La mayoría de los resultados se encuentran entre el 7 y el 9 por lo que se cumpliria el proposito del proyecto que consistía en hacer pasar un rato divertido al usuario.

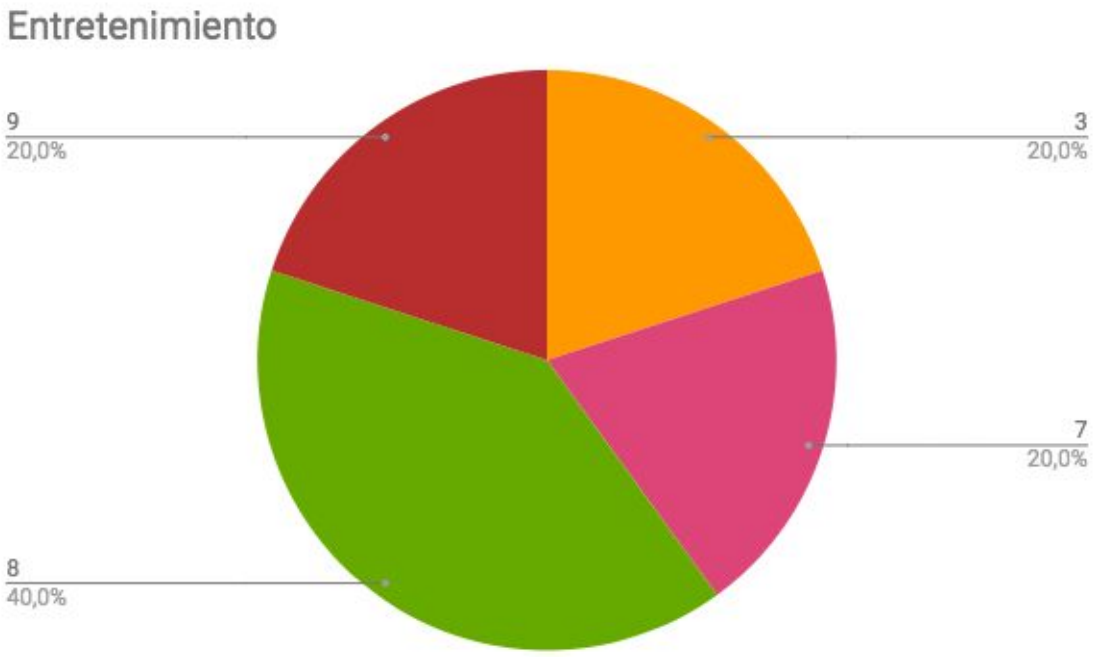

Ilustración 30 - Gráfico entretenimiento

En el apartado de lo que más ha gustado, la mayoría de los usuarios coinciden en que la dinámica de aleatoriedad de los mapas es lo mejor del juego.
Por otra parte, todos los usuarios han experimentado el mismo problema, en ciertas ocasiones hay alguna pared con texturas invisibles, es decir, la pared impide el paso del jugador, pero no se puede ver.

9 - Conclusiones y trabajo futuro

En este apartado, se explicarán de las conclusiones finales del proyecto, desde la gestión hasta un punto de vista crítico personal.

La sección con la que más desequilibrio ha habido entre tiempo estimado y tiempo real es con el aprendizaje de las herramientas. Se le tuvo que dedicar una gran cantidad de tiempo extra para poder familiarizarse con la herramienta y facilitar el desarrollo.

En cuanto a la implementación, hay una diferencia notable entre los tiempos estimados y los reales, el retraso en la optimización del movimiento del personaje y sus colisiones contra las paredes fue causado por la falta de conocimientos de la herramienta.
En las demás tareas de la implementación, el tiempo invertido ha sido más o menos el esperado, con excepción de la creación de mapas dinámicos que llevo algo más tiempo de lo estimado.

En los demás apartados el tiempo estimado, en general, ha sido parecido al tiempo estimado sin grandes diferencias..

En cuanto a la gestión de riesgos, en la sección de desarrollo, hubo un incidente con el hardware y se perdieron unas pocas horas de trabajo. El riesgo estaba contemplado y se usó una copia del backup que está en la nube(Google drive). Además de esto solo se perdieron un par de horas de trabajo en los que no se pudo trabajar.

Tiempo estimado vs real

Tarea	Tiempo estimado	Tiempo real
Captura de requisitos	25	27
Analisis	22	50
Diseño	17	15
Implementación	150	180
Gestión	7	7
Plan de pruebas	23	20
Documentación	76	85
Total	320	384

Tabla 3 - Tiempo estimado vs tiempo real

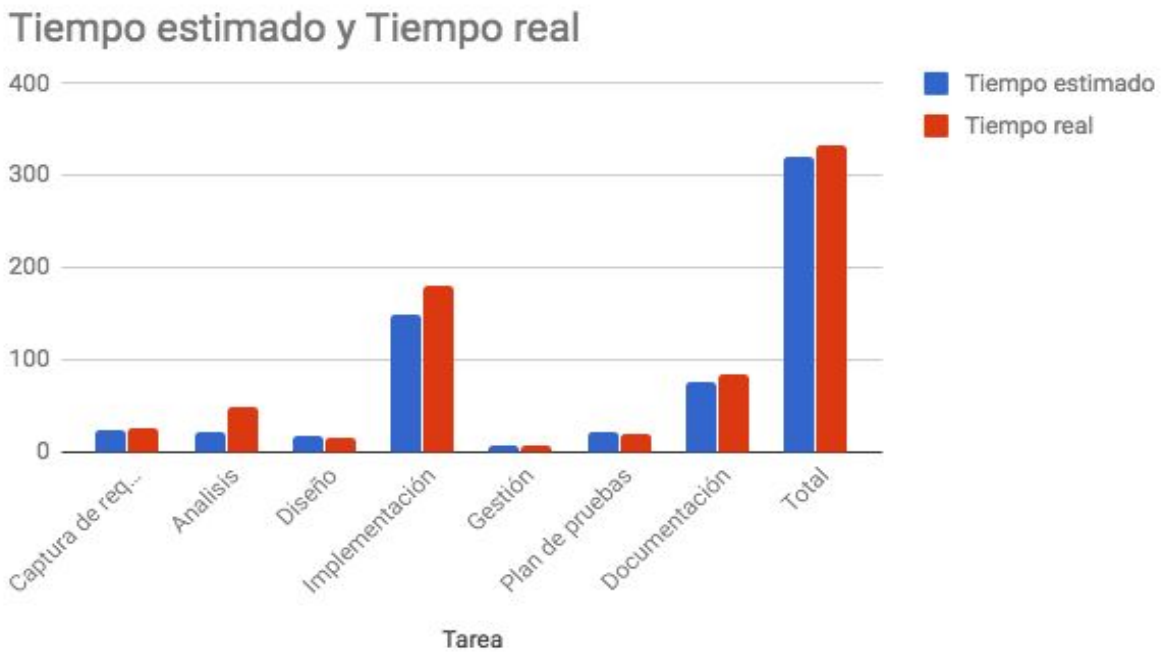

Ilustración 31 - Tiempo estimado vs tiempo real

Ya terminado el proyecto, se puede ver que los dos objetivos principales se han completado de forma satisfactoria.
Por una parte se ha aprendido a usar una tecnología nueva y un lenguaje de programación con fluidez, y por otra, se ha hecho un videojuego que puede entretener durante un corto periodo de tiempo.

A nivel personal, se han adquirido diversos conocimientos técnicos sobre la implementación de videojuegos en 2D y el lenguaje de programación C#.
En proyectos futuros, en los que requieran tecnologías nuevas, se le dedicaría más tiempo a su aprendizaje al principio del proyecto.

Por otro lado, ha sido divertido y gratificante desarrollar en Unity, ha habido momentos en los que el desconocimiento de la herramienta ha llegado a frustrar. Pero dedicándole tiempo extra se han podido evitar todos los obstáculos.
A la vez que el proyecto ha habido que compaginar estudiar y trabajar por lo que ha sido una época de mucho trabajo y de poco dormir.

En cuanto al futuro del juego, le falta mucho para llegar al nivel de un juego profesional.
Se tiene pensado añadir nuevas estéticas a los mapas, nuevos enemigos con diferentes patrones que sorprendan al jugador y le pongan las cosas aún más difíciles.
A su vez, se implementaría un modo de juego cooperativo, en el que la dificultad del juego se complica mucho más, y el jugador va a necesitar la ayuda de un compañero para poder avanzar por el juego.

Habría que dedicar más tiempo a las secciones a las que apenas se les ha dedicado tiempo, siendo la música y la parte gráfica. Bien con la ayuda de un experto o aprendiendo lo necesario para poder realizarlo sin ayuda externa.

10 - Bibliografía

Imágenes y sprites

https://commons.wikimedia.org

https://lionheart963.itch.io/

Unity & C#

Unity api

Roguelike tutorial

BoxCollier or CircleCollider

Documentación

Unity vs Unreal

Why 2D

"Why was Python created in the first place?"

C# preview

ANEXO I. DIAGRAMAS DE SECUENCIA

Disparar

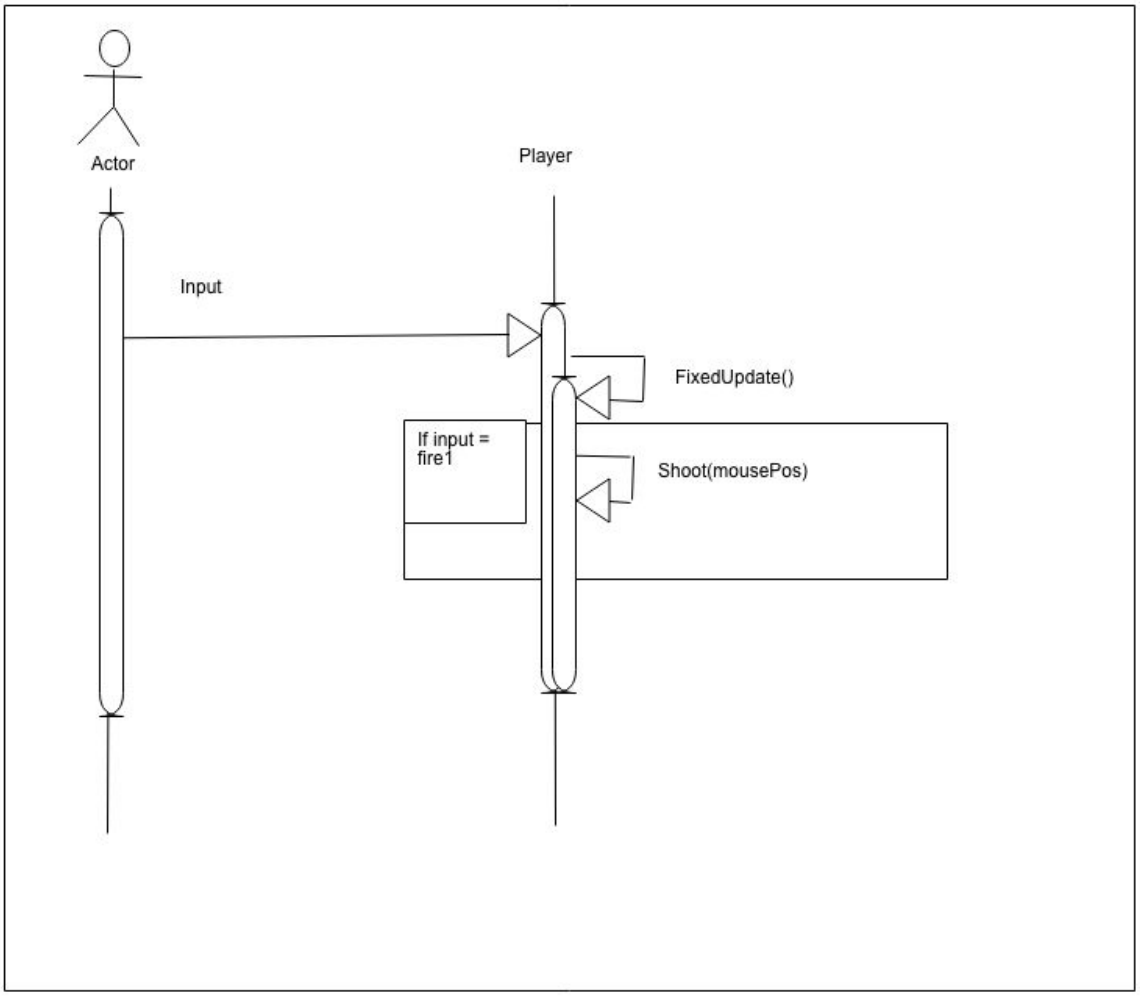

Ilustración 31 - Anexo I. Diagrama de secuencia disparar

En el momento en el que el jugador pulsa el teclado o hace click con el ratón, el juego recoge el input y en caso de que sea del click 1 del ratón, se creará un proyectil y avanzará hacia la posición en la que se encuentra actualmente el puntero del ratón.

Mover personaje

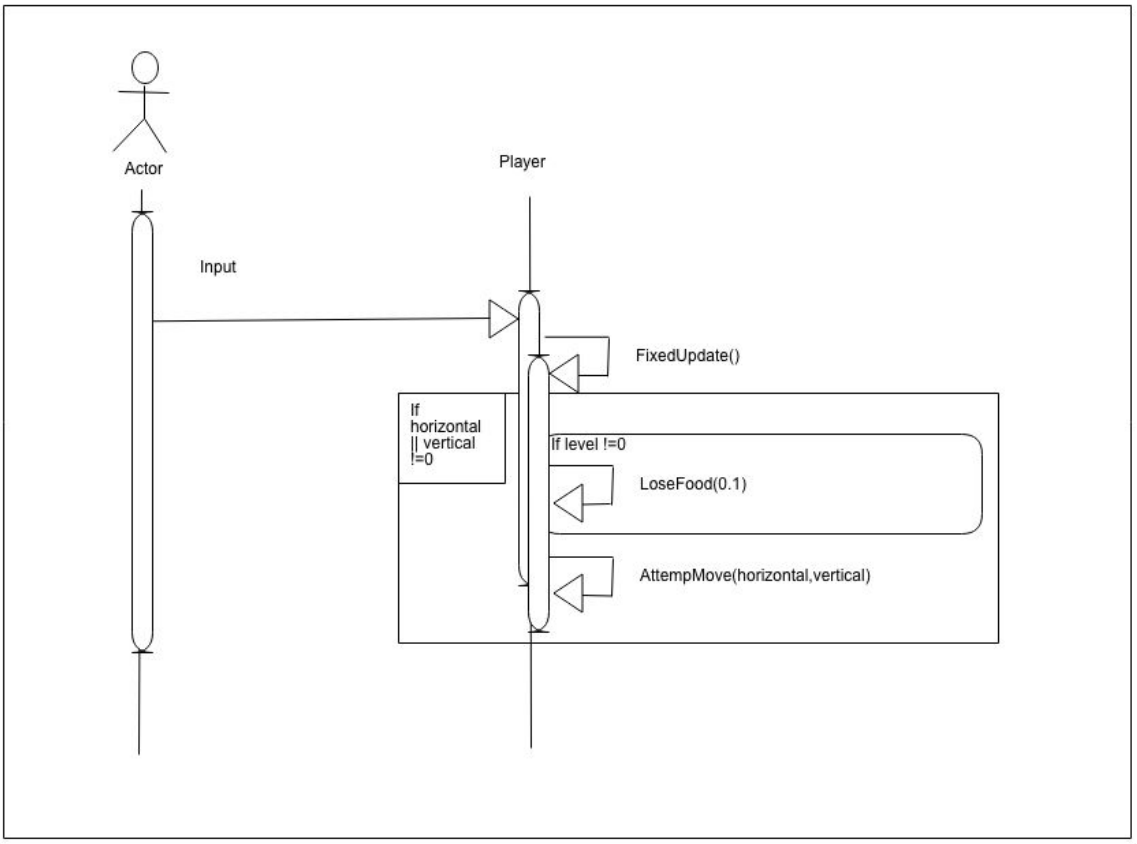

Ilustración 32 - Anexo I. Diagrama de secuencia mover personaje

En el momento en el que el jugador pulsa el teclado o hace click con el ratón, el juego recoge el input y en caso de que sea de las teclas ‘a’, ‘s’, ‘d’, ‘w’ o las flechas de dirección, este input se transforma en 2 direcciones, horizontal y vertical. Una vez se han conseguido la dirección del movimiento se le aplica al jugador, de esta forma consigue moverse de la posición a la que estaba a una en la dirección introducida por el jugador mediante el teclado.

Reiniciar nivel

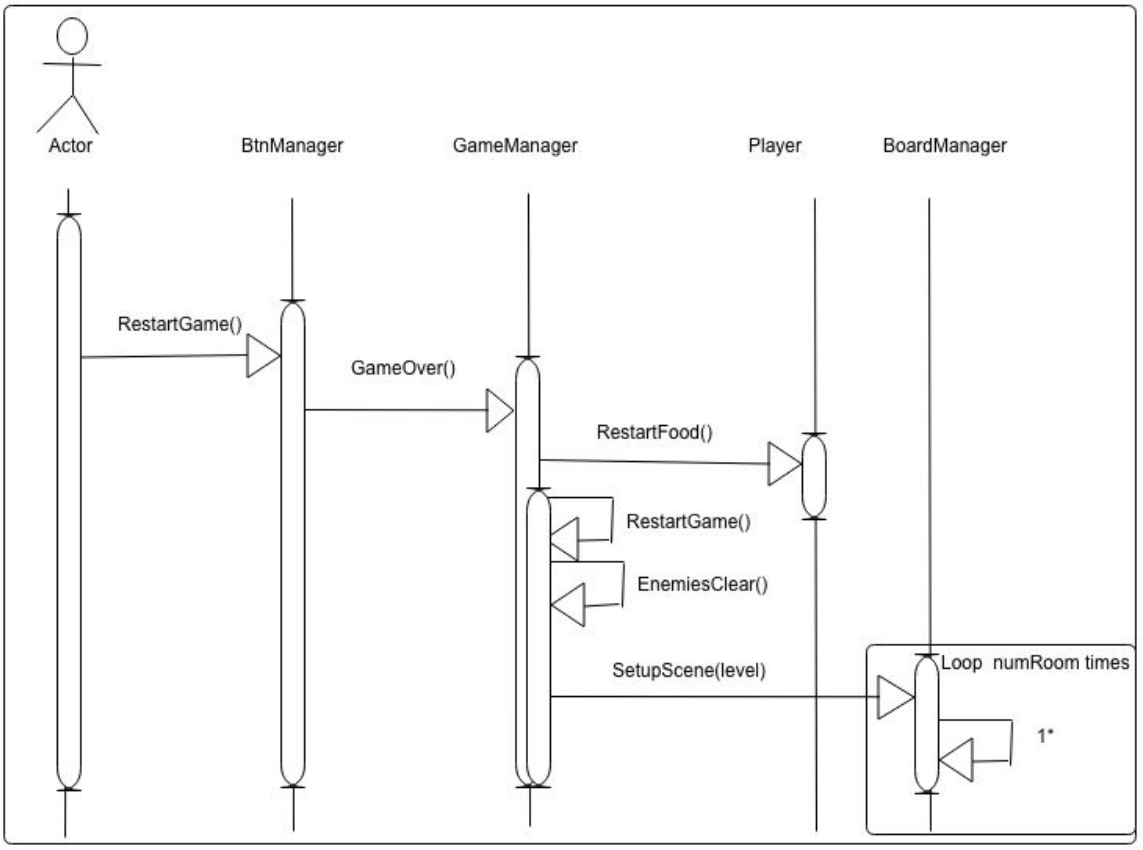

Ilustración 33 - Anexo I. Diagrama de secuencia reiniciar partida

En el momento en el que el jugador hace click en el boton de reiniciar el juego, el gestor del juego elimina todos los enemigos y reinicia el tablero.
Una vez reiniciado, se procede a volver a crear el tablero de juego llamando al gestor del tablero.

1*
Primero se coloca el suelo y las paredes mediante el método ***BoardSetup***, acto seguido inicializa una lista de posiciones dentro del mapa para colocar los objetos y enemigos usando el método ***InitiateList***. Una vez tiene la lista de posiciones disponibles, coloca los objetos y enemigos que correspondan en posiciones aleatorias mediante el método ***LayoutObjectAtRandom***.

Este proceso se repite tantas veces como salas vaya a haber en el nivel.

ANEXO II. CASOS DE USO EXTENDIDOS

Nombre: Mover jugador.

Descripción: Acción para mover al jugador por el mapa en la dirección que se quiera.

Actor: Jugador.

Precondiciones: El juego no puede estar parado ni el mapa creándose.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) Se captura el input del teclado(Flechas o ‘a’,s’,’d’ y ’w’).
2) Se transforma el input en un Vector2 de posición (Horizontal,Vertical).
3) Se aplica el Vector2 creado a la posición actual.

Postcondiciones: El jugador se ha movido de la posición en la que estaba, a otra diferente en la dirección que se le ha dicho.

Ilustración 34 - Anexo II. mover jugador posición inicial

Ilustración 35 - Anexo II. mover jugador posición final

Nombre: Disparar proyectil.

Descripción: El jugador dispara un proyectil, al hacer click con el botón izquierdo del ratón, en dirección a la posición del cursor del ratón.

Actor: Jugador.

Precondiciones: El juego no puede estar parado ni el mapa creándose.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) El jugador hace click con el botón izquierdo del ratón.
2) Se añade un proyectil a la lista de proyectiles.
3) El personaje realiza la animación de atacar
4) Se recorre la lista de proyectiles moviéndolos en dirección al cursor del ratón.

Postcondiciones: Se crea un proyectil en la posición del jugador y este avanza en la dirección actual del cursor.

Ilustración 36 - Anexo II. jugador dispara

Nombre: Reiniciar partida.

Descripción: Se reinicia la partida actual empezando desde el nivel 0.

Actor: Jugador.

Precondiciones: El juego tiene que estar parado.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) Se eliminan todos los objetos del mapa.
2) Se restaura la salud del jugador.
3) Se crea el mapa del nivel 0.

Postcondiciones: Se termina la partida actual y se empieza otra desde el inicio.

Nombre: Parar partida / reanudar partida.

Descripción: Se para el tiempo en la partida, por lo que se detiene el movimiento de todos los objetos. En caso de que ya esté parada, se reanuda como si nada hubiese pasado.

Actor: Jugador.

Precondiciones: El juego debe estar en movimiento (Parar).
El juego debe estar parado (Reanudar).

Requisitos no funcionales: No hay.

Flujo de eventos(Parar):

1) El jugador presiona la tecla ‘esc’
2) Se detiene el movimiento.

Flujo de eventos(Reanudar):

1) El jugador presiona la tecla ‘esc’ o hace click en el botón reanudar.
2) Se reanuda el movimiento.

Postcondiciones: Se cambia el tiempo de juego, bien parándolo o bien devolviendolo a la normalidad.

Nombre: Terminar partida.

Descripción: Se cierra la aplicación

Actor: Jugador.

Precondiciones: El juego debe estar parado.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) Se hace click en el botón 'Exit Game'
2) Se cierra la aplicación.

Postcondiciones: Se cierra la aplicación.

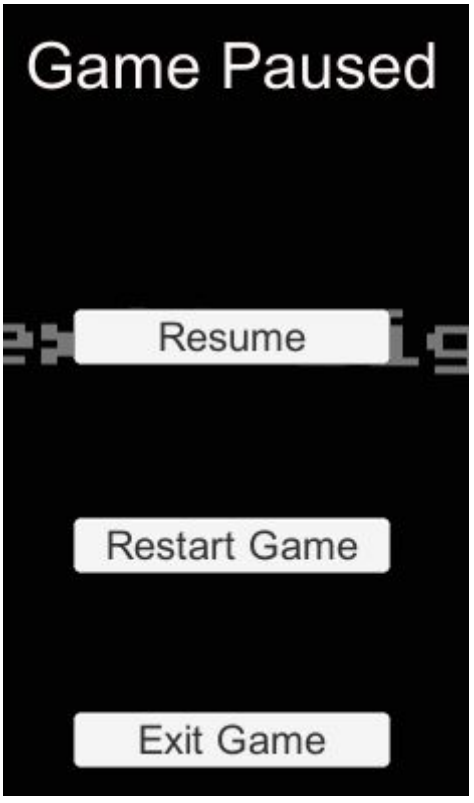

Ilustración 35 - Anexo II. menú de reanudar, reiniciar o cerrar la aplicación