Cosmic Combat

Juan Castro Albertos

Trabajo fin de Grado

Supervisado por Dr. Pablo Trinidad Martín-Arroyo

Universidad de Sevilla

junio 2024

Publicado en junio 2024 por
Juan Castro Albertos

http://www.lsi.us.es/~trinidad
ptrinidad@us.es

Yo, D. Juan Castro Albertos con NIF número 20604279M,

DECLARO

mi autoría del trabajo que se presenta en la memoria de este trabajo fin de grado que tiene por título:

Cosmic Combat

Lo cual firmo,

Fdo. D. Juan Castro Albertos
en la Universidad de Sevilla
23/06/2024

El presente proyecto lo dedico a mi familia ya que supone un gran pilar en mi vida y siempre se han preocupado por mí, dándome mucha atención y cariño.

Agradecimientos

Quiero agradecer a mis amigos Fernando y Juan Manuel por apoyar la idea de desarrollar este proyecto. En especial a mi padre por ayudarme desde siempre y motivarme a conseguir mis objetivos.

Resumen

Este Trabajo de Fin de Grado se basa en desarrollar un videojuego inspirado en los exitosos títulos *Worms* y *Crazy Planets*, siendo este último retirado del mercado aun teniendo una cantidad de jugadores considerable. Para ello, busco crear un prototipo avanzado con el potencial de poder continuarlo en el futuro y lanzarlo al mercado, aprovechando la fórmula considero no fue explotada al máximo. Como único desarrollador, busco combinar elementos estratégicos y de acción por turnos en un entorno 2D que atraiga a los jugadores. La solución se basa principalmente dentro de la plataforma *Unity* para la creación del videojuego, aprovechando su versatilidad y amplia comunidad de desarrollo.

El enfoque se seguirá mediante la metodología ágil FDD, permitiendo una planificación modular y eficiente. Esta memoria comprende un exhaustivo análisis acerca de los aspectos técnicos y de diseño sobre el videojuego. Por ello, integro apartados de estos dos ámbitos sin la necesidad de adjuntarlos por separado como un *documento de diseño de videojuegos* o *GDD* y un *documento de diseño técnico* o *TDD*. Por ello, en un apartado se detalla la mecánica de juego, personajes, armas y niveles, asegurando así una base sólida para la implementación. En el otro, se incorporan los aspectos técnicos, decisiones de diseño, gestión de cambios gestión de las físicas y los efectos visuales.

Como objetivo no solo busco presentar un producto jugable, sino también proporcionar un estudio detallado de la metodología y técnicas utilizadas en el proceso de desarrollo, con la visión de que esta experiencia académica sea un paso importante hacia un proyecto exitoso en el mercado de los videojuegos.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Introducción

Contexto

La llegada de los videojuegos a la industria del entretenimiento ha sido boom que ha llenado las portadas de todos los medios económicos por su fuerza [...]. Los videojuegos representaron el 42,1 % de los ingresos totales del entretenimiento en 2022, creciendo en 2,3 % respecto a 2021.

The Objective,
Periódico digital

Los videojuegos han supuesto un antes y después en la historia contemporánea, siendo uno de los máximos exponentes del entretenimiento. Por ello, vamos a ver su trayectoria y qué aspectos han ayudado a moldear la idea de la que parte este proyecto.

1.1 El mundo del videojuego

La industria del videojuego es una de las industrias más llamativas y rentables. A día de hoy, está experimentando un crecimiento acelerado y parece que no va a frenarse pronto. Según el periódico digital *The Objective* [18], esta industria ya genera más dinero que la música y el cine juntos. Y es que desde los años 70 con la llegada de videoconsolas como la *Odyssey* o la *Atari 2600* [21] esta industria no ha dejado de estar en auge, sobre todo entre los más jóvenes, ¿y en los no tan jóvenes?

A través de los datos obtenidos y analizados por el portal estadístico Statista [17], se hizo un estudio por edad y sexo de consumidores de videojuegos en España en 2022. Este muestra cómo el número de encuestados en cada rango de edades adultas duplica a los segmentos de edades no adultas.

1.2 Juegos en línea dentro de Facebook

Fijándonos en este público adulto más numeroso y remontándonos a 2010, surgieron videojuegos en línea en plataformas como Facebook. En esta red social algo limitada para jóvenes por las restricciones de edad, era necesario tener una cuenta para poder acceder a estos juegos. Y es que varios de estos juegos aun no siendo tan accesibles, eran todo un éxito y motivaban crear una cuenta para poder disfrutarlos. Al estar acotados a una comunidad tan grande como Facebook, encontrar jugadores siempre era posible. Lo interesante era invitar a jugar a los amigos que uno tuviera registrados dentro de la plataforma ya que permitía desbloquear ciertas ventajas y a su vez crecer el número de jugadores dentro de una comunidad. De hecho, bastantes videojuegos del presente usaron esta plataforma para ganar popularidad, como *City Skylines* o *Angry Birds*.

1.3 El auge de los juegos de estrategia

Dentro de este marco, nos centramos en juegos en línea dentro de Facebook como *Wild Ones* y *Crazy Planets*, siendo este último mi mayor inspiración y de donde procede la idea original de este proyecto. Ambos juegos fueron eliminados de la plataforma aun cuando su público era abundante y estaba en crecimiento ya que Facebook se estaba haciendo bastante popular. *Wild Ones* era a grandes rasgos, una copia de *Worms* que ofrecía un sistema de microtransacciones y opciones de personalización que para aque-

lla época, eran un adelanto a lo que estamos acostumbrados a ver en los juegos de la actualidad. Por otro lado, *Crazy Planets* ofrecía una experiencia parecida, solo que cambiaba la fórmula del *Player versus Player* (PvP) por un *Player versus Enviroment* (PvE). La gracia era superar los distintos niveles donde la dificultad aumentaba y las microtransacciones conjunto con tener amigos en la plataforma que jugaran activamente servía de ayuda para completar el juego. En el mercado podemos encontrar multitud de juegos con estas características; sin embargo este ofrecía una experiencia distinta al tener un terreno esférico, donde apuntar y acertar al objetivo era más complicado.

Por ello ante el problema de que estos juegos no siguen en el mercado y tuvieron una buena trayectoria, decidí crear un prototipo que tuviera muchos aspectos de valor de estos 2 juegos. Con ello volvería a rescatar un género con buen potencial y así captar tanto un público que disfrutó este género como nuevos jugadores que nunca han tenido la oportunidad de probar los juegos antes descritos.

1.4 Estado del arte

En cuanto a números, podemos afirmar que la industria del videojuego es de las más rentables actualmente. Fijándonos en el anterior artículo expuesto por *The Objective*, en 2022 a nivel mundial generó 175.800 millones de dólares, previendo que se generen 218.700 millones en 2024. Estos números superan incluso a la industria del libro individualmente (120,1 mil millones).

Si nos fijamos a nivel tecnológico, es lógico pensar que detrás de tantos números existe una gran apuesta por la tecnología, en investigación y desarrollo. Y es que en esta década estamos viendo estos avances con paradigmas como la *Realidad Virtual*, los gráficos realistas o la tan famosa *Inteligencia Artificial*. La llegada de motores como *Unreal Engine 5* que traen toda esta lógica implementada hacen posible el desarrollo de proyectos muy detallados de manera gratuita y al alcance de cualquiera.

Objetivos

Concretamos qué queremos conseguir con el desarrollo de este proyecto. Definimos los límites y necesidades que debemos cubrir para que sea un producto competitivo y considerable.

2.1 Motivación

Este proyecto resulta en un programa cuyo *software* pueda ser incluido en un futuro en plataformas de videojuegos como por ejemplo *Steam*, sobre todo si se extiende en contenido y *QoL*. De este prototipo de juego se pide que sea innovador frente a los juegos populares pues comparte un género que triunfó hace unos años. Para ello podemos fijarnos en los múltiples juegos del título *Worms*.

La causa de decantarme por este juego parte de la idea de que es un género consistente y que no ha sabido aprovecharse bien durante el tiempo, dejando así al consumidor poco catálogo de juegos en la actualidad

Si tenemos en cuenta que este juego puede lanzarse en plataformas de compraventa de videojuegos como la mencionada anteriormente, la popularidad y valor de este producto serían mayores al aumentar su visibilidad y disponibilidad. El público objetivo no es ningún perfil muy concreto, simplemente un jugador de videojuegos casual que puede recrear parte de su tiempo en un juego arcade.

Además existe un público concreto que se inclina por géneros de plataformas o si se extiende para tener un apartado multijugador, tendría un componente competitivo del género *PvP*.

2.2 Listado de objetivos

Objetivo 1. Introducción novedosa al mercado: En esta fase buscamos que se consiga un público de jugadores que apoyen el juego. También se busca una base de críticas constructivas para ver una tendencia en las respuestas por parte de la comunidad.

Objetivo 2. Popularizar el género: Se busca revivir el auge de juegos de estrategia competitiva en plataformas dentro del mercado de videojuegos, consolidando y acotando una comunidad de consumidores como posibles clientes.

Objetivo 3. Aprender en base al proyecto: No es fácil convencer a un inversor para que recurra a rentabilizar la idea. Por lo que quiero aprovechar que tengo una visión optimista y ganas de aprender para avanzar en este campo y poder así aspirar a proyectos más grandes.

Parte II
Organización del proyecto

Metodología

Veremos cómo voy a organizar este proyecto y qué tipo de metodología voy a emplear. Entraré en detalle sobre las fases en la que se compone y la manera en la que lo voy a adaptar para trabajar.

3.1 Estructura organizacional del proyecto

La organización del proyecto es simple ya que al ser individual, todo recae sobre una persona. Los aspectos del proyecto se resumen en: desarrollo (código, diseño gráfico, de niveles, etc), *testing*, documentación y otros aspectos como la ejecución de la metodología y la gestión del repositorio.

Además, me he inspirado entre otros [2] [7] [8] en el manual *PMBOKv6* [16] para estructurar y redactar cada apartado de la manera más completa posible.

3.2 Metodología de desarrollo

3.2.1 Metodologías ágiles: FDD

Para el desarrollo, he basado la organización y ejecución en la metodología FDD [4]. Esta metodología ágil se basa en desarrollar *software* partiendo de una idea genérica que se va dividiendo en pequeñas partes. Estas pequeñas partes se llaman funcionalidades y se van incluyendo secuencialmente hasta completar el proyecto esperado. Cada funcionalidad no puede ser tan compleja que ocupe más de 2 semanas en completarse, por lo que en casos así se dividen en funcionalidades más simples. Con todo esto, una funcionalidad debe ser precisada con varios requisitos a implementar en forma de tareas las cuales sirven de índice para el desarrollador y le permite desglosar cada funcionalidad para tenerlas más controladas a la hora de probar o rectificar partes del código.

3.2.2 Política de *Commits*

A medida que iba actualizando y completando el proyecto desde que lo creé, subía los cambios poco a poco a Git. Por ello decidí crear una política de mensajes que adjuntar en cada *commit* o conjunto de cambios. Existen 4 tipos de *commits* dentro del registro de cambios:

1. **Proyecto**: Son los relativos a la sincronización del proyecto en Git e inserción de elementos necesarios para el desarrollo del proyecto, como imágenes o *sprites*.

2. **Documentación**: Estos ocupan el desarrollo de este documento y elementos relativos al mismo, como el diagrama UML o tablas y figuras previamente insertadas.

3. **Desarrollo**: Este conjunto de *commits* es amplio ya que ocupa la mayoría de horas y contenido del proyecto. Solo en este tipo de cambios decidí insertar tras el cuerpo del mensaje 2 saltos de línea y escribir un resumen corto en el *commit* con información relevante para filtrar información durante el desarrollo posibles errores.

4. **Testing**: A través de estos commits, realizaba pruebas de jugabilidad sobre el prototipo para ver si era adecuado hasta ese punto de desarrollo.

Concretamente, los mensajes de *commit* los redactaba de la siguiente manera:

[TIPO DE COMMIT]: Mensaje de Commit establecido previo desarrollo

(Breve resumen de menos de 30 palabras si [TIPO DE COMMIT] = 'DESARROLLO')"

3.2.3 Fases de la metodología

Para llevar a cabo todo el desarrollo del *software* hay que seguir un total de 5 pasos:

1. Desarrollar el modelo general: se define el alcance y el modelo general de la solución.

2. Crear lista de funcionalidades generales: se segmenta el modelo general en funcionalidades las cuales se acotan a su vez en acción, resultado y objeto.

3. Concebir el plan para cada funcionalidad: se determina el orden de desarrollo para cada funcionalidad con los pros y contras en la ejecución. Además se asignan a los programadores las primeras funcionalidades.

4. Realizar un diseño por funcionalidad: el programador determina las prioridades y forma de ejecutar la funcionalidad en esas 2 semanas. El equipo le da el visto bueno al diseño antes de que se empiece.

5. Ejecutar y probar la funcionalidad: se implementan todos los pasos y elementos necesarios para completar la funcionalidad. A continuación se integra y prueba en el sistema, dando así por terminada la tarea.

3.2.4 Adaptación de la metodología

En este proyecto he adaptado la metodología ya que el *software* esperado es un videojuego y podía tanto omitir detalles que no proceden como emplear algunas buenas prácticas de desarrollo.

En primer lugar, cabe destacar que a medida que se completa una funcionalidad, esta quedaba totalmente funcional y de manera completa a menos que hubiese algún elemento adicional que mantuviese relación con una funcionalidad a desarrollar en el futuro. Por ello, decidí no implementar elementos incompletos aunque completaran más la funcionalidad a nivel global y por adelantar trabajo.

Otro elemento a tener en cuenta dentro del desarrollo de *Unity* es el no utilizar métodos para crear instacias de elementos nuevos y destruyéndolos desde el código. En otras palabras, reciclar entidades dentro del juego [19].

Planificación

Primero de todo, vamos a ver cómo voy a desglosar el desarrollo del proyecto en un número reducido de iteraciones y evaluar aspectos como la carga de trabajo y la periodicidad de las revisiones.

4.1 Resumen temporal del proyecto

Resumen del proyecto	
Fecha de inicio	06/08/2023
Fecha de fin	24/06/2024
Periodicidad de las revisiones	2 meses
Carga de trabajo semanal	12 horas
Horas totales previstas	300 horas
Horas finales	293 horas

Cuadro 4.1: Tabla resumen de tiempos y planificación

4.2 Planificación inicial

Aquí un desglose de las iteraciones, comienzo y fin de cada una:

Resumen de iteraciones	
Iteración 1	6/08/23 a 23/08/23
Iteración 2	10/10/23 a 10/12/23
Iteración 3	4/02/24 a 24/06/24

Cuadro 4.2: Planificación temporal de iteraciones

He decidido que la 1º iteración durará en torno a 2 semanas, atendiendo al tiempo del que quiero disponer previo a la preparación de otros exámenes. En ella, busco familiarizarme con el lenguaje LaTeX y repasar conceptos de *Unity*.

La 2ª segunda iteración comenzará sobre mediados de Noviembre y abarcará hasta las vacaciones de Navidad. Aquí es donde empezará el desarrollo del proyecto dentro de *Unity* y se añadirán algunas características a implementar.

Por último, la tercera iteración servirá para implementar las funciones restantes y cerrar el proyecto. Esta etapa es donde se encuentra la mayoría del contenido y aunque la carga de trabajo ha sido irregularmente repartida, el ritmo de trabajo lo marcaba la gestión que tenía sobre asuntos académicos, laborales y personales.

Costes

Definimos los costes del proyecto en base a los recursos de los que hemos hecho uso durante el desarrollo. Los costes se van a definir de la manera más exacta posible.

5.1 Resumen de costes del proyecto

Resumen del proyecto	
Costes de personal	7.729,72 €
Sueldo bruto	5.797,29 €
Costes sociales	1.932,43 €
Costes materiales	192,75 €
Costes indirectos	119,53 €
TOTAL	8.042 €

Cuadro 5.1: Tabla resumen de costes

5.2 Costes de personal

Después de hacer varias consultas sobre ofertas de empleo de programador en *Unity* en portales como *LinkedIn*, definí de las siguientes estimaciones el coste por hora sobre los tipos de tareas que involucran el proyecto:

1. **Desarrollador Back-end en *Unity*** (80.000-150.000 USD): 42 €/hora.
2. **Desarrollador Front-end en *Unity*** (60.000-120.000 USD): 30 €/hora.
3. **Diseñador UX/UI en *Unity*** (70.000-140.000 USD): 36 €/hora.
4. **Tester en *Unity*** (40.000-90.000 USD): 20 €/hora.

Vamos a calcular los costes de cada rol del proyecto ya que en el recuento de horas en *Clockify* solo figuro yo como único desarrollador.

1. Coste como desarrollador *Back-end* en *Unity*: 141,22 horas * 42 €/hora = 5.931,24 €.
2. Coste como desarrollador *Front-end* en *Unity*: 51.77 horas * 30 €/hora = 1553.10 €.
3. Coste como diseñador *UX/UI* en *Unity*: 3 horas * 36 €/hora = 108 €.
4. Coste como *Tester* en *Unity*: 6,87 horas * 20€/hora = 137,38 €.

Los **Costes Sociales** son del 25% sobre el total de **Costes de personal**, aunque hay que tener en cuenta que de *Sueldro bruto* hay que seguir descontando antes de darle la parte correspondiente al desarrollador.

5.3 Costes materiales

Para este apartado destacamos que la membresía por el uso de *Unity Personal* que es la versión usada para el desarrollo es gratuita. Sin embargo, este proyecto necesitaría una versión distinta para poder publicarlo en plataformas de videojuegos. Por lo que el coste de la membresía de *Unity Pro* sería de 172,75€ a razón de pagar un mes para poder publicarlo, entre otras ventajas.

Por otro lado, debemos sumar el coste del diseño en *Pixel Art* de los diseños usados para entidades dentro del juego. Teniendo en cuenta que he depositado entorno a 4 horas buscando recursos gratuitos o bien creándolos, a un coste de 5 € por hora son 20 € por este apartado gráfico.

5.4 Costes indirectos

Solo he decidido contemplar la amortización de mi equipo, el coste de la luz y el porcentaje equivalente al alquiler del tiempo de desarrollo.

De 4 años de antigüedad que tiene mi equipo a un coste de 750€, resulta en un coste de amortización en 293 horas totales de 1,72 €.

Amortización PC últimas 293 horas = 750 € (4 años * 416 horas laborables/año)

En cuanto a luz gastada por el equipo, atendiendo a el coste promedio del kWh en Sevilla es de 0.26305 €. Por tanto en 293 horas haría un total de **1.86 €**.

Por último desde que empecé el proyecto hasta que lo terminé han transcurrido 219 días. Hallándo el cociente entre horas depositadas y días dedicados y multiplicado por 260€ /mes resulta en:

293/5.256 horas * 8 meses * 260 €/mes = **115.95 €**.

Parte III

Desarrollo del proyecto

Arranque

Para llevar a cabo este proyecto, ha sido necesario un proceso de estudio del entorno, un análisis de los elementos a implementar y el desarrollo como producto de la idea. En esta parte vamos a ver en profundidad este último punto.

6.1 Lista de características

El proyecto parte de crear un prototipo basado en *Wild Ones*, con algunas mecánicas extraídas de *Crazy Planets* para convertirlo en un juego de estrategia por turnos. La escasez de juegos de este estilo hace que sea una buena oportunidad sacar un proyecto similiar al mercado pues volvería a reinventar el género que en su día *Worms* hizo que se jugara tanto y estuviera en boca de muchos. El juego cuenta con dos modos de juego, uno enfocado a un *PvE* cooperativo donde los jugadores se enfrentan a los personajes creados por la máquina. El otro modo de juego es similar solo que los jugadores también luchan entre sí (*PvPvE*).

De esta idea inicial surge un conjunto de características agrupadas en distintos focos de desarrollo para alcanzar la meta propuesta y obtener el producto al que aspiro. Estas características son:

Desarrollo a nivel de diseño de la jugabilidad:

- El Planeta.
- Los Personajes.
- Los Monstruos.
- Los Proyectiles y Armas.
- El sistema de trazado y trayectorias.
- Un inventario de armas para cada personaje.
- El sistema de acción por turnos.
- Configuración de la IA de Monstruos.
- El Cohete.
- La gestión de partidas.

Desarrollo a nivel de interfaz de usuario:

- Menús de juego.
- Una base de datos en la nube.

- Un formulario de inicio de sesión y registro.
- Sistema de puntuación y tienda.

Desarrollo a nivel de arte y ambientación:

- Música ambiental y sonidos.
- Animación de efectos.

6.2 Diseño arquitectónico

6.2.1 Sistemas de preproducción

El juego consiste en un espacio en 2 dimensiones donde podemos controlar un personaje en un planeta. En él, aparecen monstruos que debemos eliminar con las diferentes armas que a su vez irán apareciendo por todo el planeta. Una vez hayamos vencido a 10 monstruos aparecerá el "Jefe de los Monstruos", uno mucho más difícil pero obligatorio de vencer. Cuando sea derrotado, aparecerá un cohete en una parte del planeta y tendremos que llegar a él para terminar la partida.

6.2.2 Sistemas de producción

Diseño del juego

Como he mencionado antes, existen 2 modos de juego. En el modo **Cooperativo**, existen 3 personajes que cooperan para derrotar a los monstruos y finalizan la partida cuando agotan las 3 vidas que tienen y todos son derrotados o bien cuando todos llegan al cohete.

En el modo **Todos contra Todos** estos 3 personajes también luchan entre sí para ver quién derrota al Jefe de los Monstruos. Solamente hay un ganador y es el último que quede vivo o bien quien derrote a este. Aquí cada personaje tiene 3 vidas y en ningún momento aparece el cohete.

Diseño artístico

A la hora del diseño de los personajes, monstruos y la estética de cada partida, he usado recursos gratuitos de algunos sitios *web*. Para crear los 4 personajes y 4 mons-

truos he usado *Creature Mixer* [11]. Existen 3 planetas creados con *Pixel Planet Generator* [9] y un fondo estrellado con acabado pixelado [10].

Para incluir música y efectos de sonido en el juego, he escogido pistas de audio sin derechos de autor en [1], una web en la que autores suben sus pistas de audio para que quien quiera puedas usarlas. Por ello elegí pistas de audio cortas para los sonidos a la hora de disparar un proyectil, pulsar un botón, etc. Y para la música del juego elegí una pista de audio aún más larga que se ejecuta mientras el juego esté siendo ejecutado.

Si bien es verdad que con *UIToolkit*, la nueva función de *Unity* para hacer el canvas y la exposición de elementos gráficos personalizados de manera más sencilla a mí me resultó más complejo. Esto es debido a que al ser un sistema novedoso, existe poca documentación y pocos tutoriales.

La otra manera y de la que tenía conocimiento sobre como hacer lo anterior mencionado es la creación de elementos gráficos y la situación de un canvas entre la cámara y el juego. Esta herramienta permite la creación del HUD del videojuego de manera más cómoda, permitiendo además un mayor acoplamiento entre el código y la gestión de la interfaz visual.

El juego no cuenta con ningún modo de juego con cinemáticas que narren una historia pues la jugabilidad se centra en ir ganando partidas y sumando puntos.

Diseño mecánico

En este prototipo de videojuego no se ha desarrollado un sistema de juego "*on line*". En ambos modos de juego, los personajes son controlados por un solo jugador ya que implementar un sistema multijugador requeriría más horas de las previstas para este proyecto. Sin embargo, todos los monstruos son dirigidos por la máquina, obedeciendo una serie de condiciones booleanas.

Motor del juego

La herramienta que uso para crear este videojuego es *Unity* [20]. He elegido este motor de videojuegos por varias razones. Entre ellas la más relevante es que *Unity* es un motor de videojuegos fácil de usar pues la curva de aprendizaje conjunto con el desarrollo de cualquier proyecto es menor que con otras herramientas del mercado. Además cuenta con variedad de tutoriales y recursos gratuitos para no tener que crear cada componente desde cero. A mi favor, cuento con experiencia previa en el uso de esta herramienta lo cual me sirve para depositar menos tiempo en aprender sobre esta tecnología.

Dentro de ella, emplearé *Rigidbody* como componenete simulador de físicas ya que es la alternativa primordial y más simple a la hora de añadir físicas. Tanto el planeta como las entidades dentro del juego obedecerán este sistema de físicas.

La IA que implemento a los personajes del modo Todos contra Todos y monstruos es sencilla: en el turno del monstruo, este camina aleatoriamente hacia una dirección durante 10 segundos. Acto seguido, en el caso del Jefe de los Monstruos, este evalúa la distancia a la que está cada personaje y puede disparar su Rayo Láser a uno al azar o bien fallar. Los demás monstruos disparan un Misil hacia una dirección totalmente impredecible y aleatoria.

Diseño Técnico

Al ser este un prototipo de un videojuego, no está completo al nivel de un juego puesto en el mercado. Por ello, la idea de este proyecto es solo llevada a la plataforma en la que está desarrollado, es decir, a PC. No obstante, sería interesante en un futuro llevar este juego completamente desarrollado a plataformas portátiles ya que no ofrece grandes demandas de recursos ni exige un nivel de habilidad considerable.

6.2.3 Pruebas

A la hora de probar el juego, conforme que iba desarrollando nuevas funcionalidades, conseguía imprimir por pantalla valores de forma temporal o visualizar el efecto de esa implementación en pantalla por medio de pruebas cortas de juego a fin de corroborar que todo iba según lo planeado. Así también tenía una visión global de cómo me sentía como jugador y ya no solo como desarrollador, por lo que iba cambiando constantes y detalles hasta conseguir el resultado esperado.

No obstante, en cuanto a código la parte de pruebas he hecho que fuese corta y donde solo se comprueben valores nominales para ver cómo trabajan las funciones que lo forman. De modo que la etapa de pruebas ha sido cubierta con pruebas unitarias con un total de 4 pruebas formales incluidas como última implementación al final del desarrollo del proyecto. Me he llevado más tiempo del esperado descubrir cómo implementar solo pruebas unitarias al proyecto, por lo que he descartado en esta fase el realizar pruebas funcionales donde se ejecute el juego. Considero que hacer un conjunto de pruebas tan avanzado no encajaría dentro del alcance de un proyecto como este.

Iteración 1

Primero de todo, vamos a ver qué características iniciales vamos a implementar dentro del proyecto, con su duración estimada y cómo las vamos a desarrollar. Luego, se hará una estimación temporal en cada iteración.

7.1 Características a desarrollar

1. El Planeta - 6 horas

1. **Ajuste cámara y fondo de pantalla del juego** (1 hora): insertar el archivo *PNG* de fondo de pantalla y enfocar la cámara a dicho PNG para que se vea como fondo de la escena, sin emplear ningún componente de luz.
2. **Insertar el planeta y los tipos de planeta** (1 hora): colocar el *PNG* del Planeta en el centro de la cámara y crear un fichero para albergar las texturas de Planeta durante la selección de juego.
3. **Añadir el sistema de colisiones al Planeta** (1 hora): delimitar el área de colisión del Planeta y acotarlo dentro de la cámara de manera fija para ajustarlo dentro de la pantalla durante las partidas.
4. **Generación de puntos de aparición** (2 horas): crear una función donde se puedan generar coordenadas en la superficie del Planeta que sirvan como puntos de aparición para Personajes, Monstruos y Munición de Armas.

2. Los Personajes - 7 horas

1. **Crear entidades Municion y Arma** (1 hora): para que el Personaje además de recoger Munición pueda usar armas conjunto con los Monstruos, es necesario crear ambas entidades.
2. **Insertar la entidad de Personaje jugable y los tipos de Personaje** (2 horas): crear la entidad Personaje, colocar un Personaje controlable en la escena que se vea atraído hacia el Planeta con un componente de Gravedad.
3. **Añadir movimiento del Personaje y su sistema de colisiones** (4 horas): hacer que el Personaje jugable se pueda mover alrededor del Planeta y su apariencia se alinee con el Planeta, colocando sus pies más cerca de la superficie que el torso.

3. Los Monstruos - 4 horas

1. **Insertar la entidad de Monstruo y los tipos de Monstruo** (2 horas): crear la entidad Monstruo, colocarlo en la escena que se vea atraído hacia el Planeta con un componente de Gravedad.

2. **Movimiento de los Monstruos y sus sistemas de colisiones** (2 horas): hacer que el Monstruo se mueva aleatoriamente alrededor del Planeta y su apariencia se alinee con el Planeta, colocando sus pies más cerca de la superficie que el torso.

7.2 Diseño

Los elementos visuales del proyecto han sido diseñados por mí con ciertas herramientas o bien tomadas de Internet. Por ello he usado una diversidad de diseños que calzan lo mejor posible con la idea que tenía al concebirlo. [9] [3] [15] [13] [12] [5]

El diseño inicial del proyecto parte del primer diagrama UML Fig. §7.1 que cree:

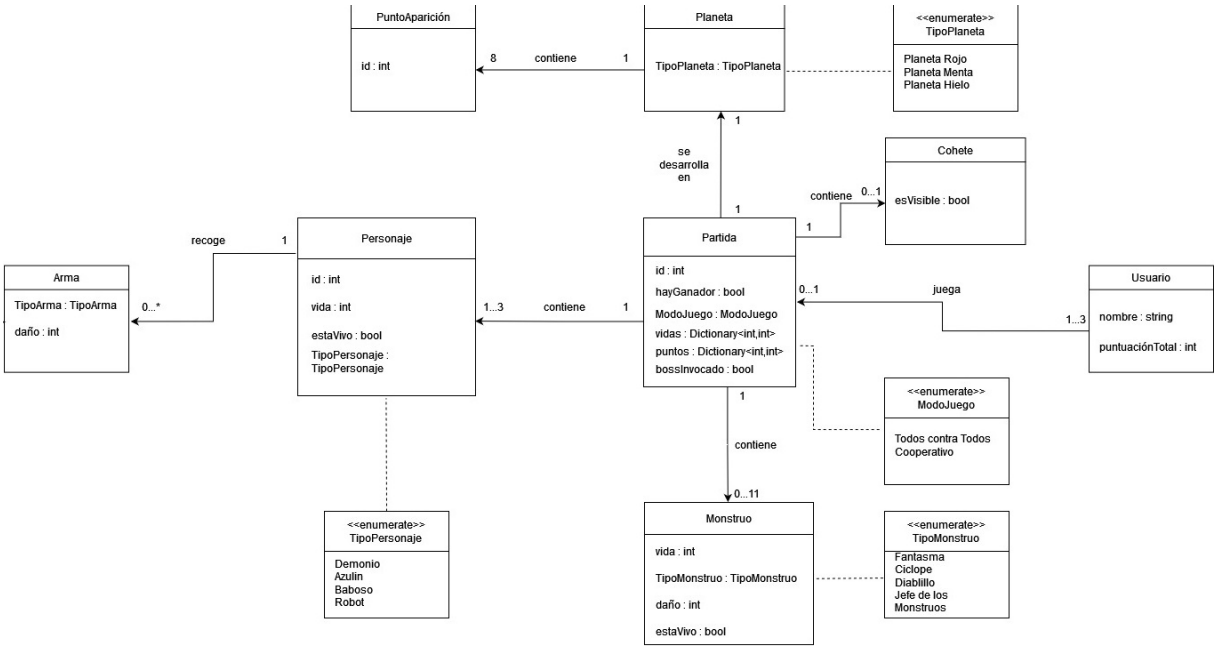

Figura 7.1: Diagrama UML de diseño: estructura del juego

En un primer momento, pensaba crear puntos fijos durante la partida para que los Personajes, Monstruos y Munición aparecieran separados entre sí. Más tarde mientras lo desarrollaba, me di cuenta que tenía el mismo sentido crear coordenadas aleatorias y hacer aparecer las entidades en esos puntos.

Por tanto, como quería que las entidades se generaran alrededor del contorno del planeta decidí usar el teorema de Pitágoras: $a^2+b^2=c^2$. Supuse que el radio del planeta al cuadrado (c) era equivalente a la suma de los cuadrados de 2 coordenadas [a,b] siendo el [0,0, 0] el centro del planeta. Después de aplicar este cambio, ya no era necesario

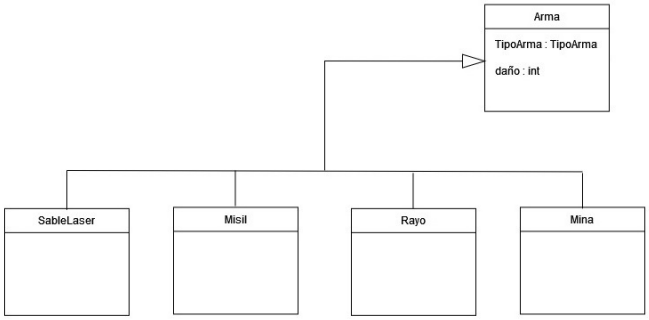

Figura 7.2: Diagrama UML de diseño: tipos de armas

incluir una entidad de punto de aparición y el diagrama UML Fig. §7.3 variaba así:

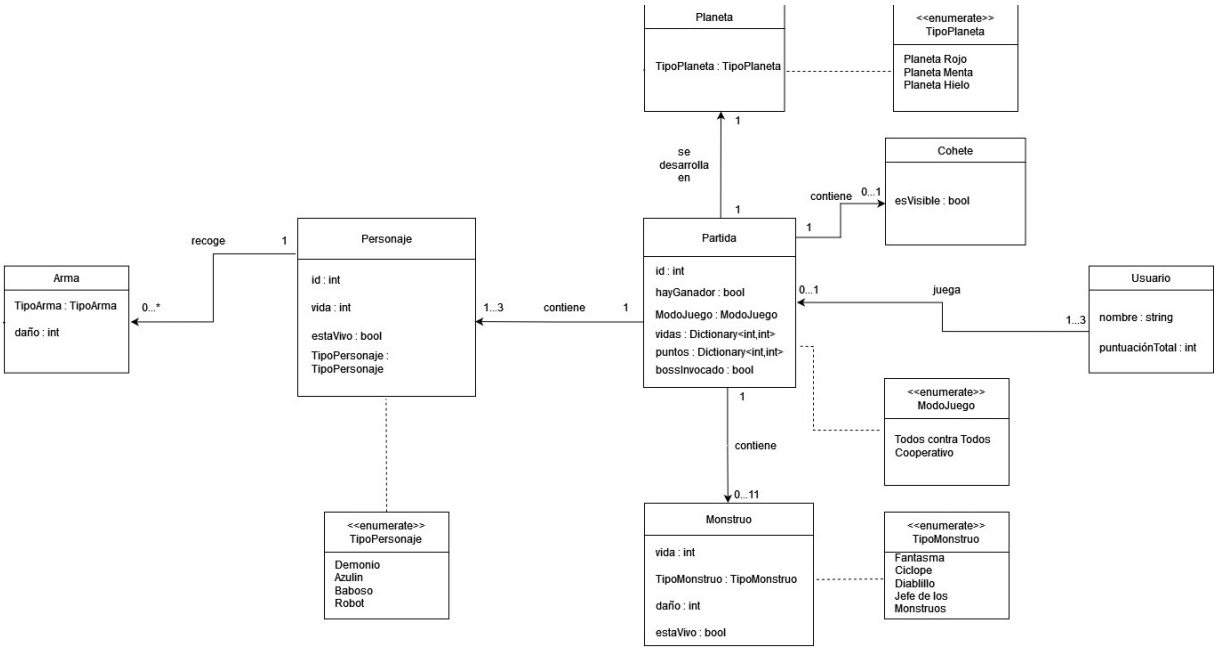

Figura 7.3: Diagrama UML de diseño: eliminación entidad Punto de Aparición

Durante el desarrollo de la entidad Personaje llegué a la conclusión de que la separación de las tareas a implementar dentro de la funcionalidad era incorrecta. En lugar de crear la entidad sin su movimiento con las máquinas de estado de las animaciones, vi más conveniente crear por un lado la entidad completa con las entidades de las que dependía. Esto se debió a que fui creando y refactorizando partes del código que eran necesarias por lo que para que no hubiese tanta diferencia en líneas de código entre distintas etapas de la funcionalidad. Por otro lado, podía así diferenciar las partes principales de la entidad y no dejar una etapa con varias partes sin completar.

Memorando técnico 0001	
Asunto	Generación entidades entorno al Planeta.
Resumen	Cálculo con teorema de Pitágoras.
Factores causantes	Se desea poder generar coordenadas cerca del perímetro del Planeta y que haya una distancia constante equivalente al radio de la circunferencia que forma el Planeta.
Solución	Se considera el radio del Planeta como la hipotenusa de un triángulo rectángulo imaginario. Luego se genera de manera aleatoria un cateto de dicho triángulo promediado entre 0 y el radio del Planeta . Por medio del teorema de Pitágoras, se generan las coordenadas X e Y.
Motivación	Se propone como solución eficiente para resolver un problema matemático.
Alternativas	Se puede calcular repetidamente coordendas al azar hasta encontrar una cuya distancia hacia el centro de la circunferencia del Planeta resulte en el radio del mismo.

Cuadro 7.1: Memorando técnico 0001

Después de este cambio, surgió el primer desvío del tiempo previsto en el proyecto ya que empleé 2 veces más del tiempo planificado en comprobar cada una de las características que debía cumplir la entidad Personaje y la gravedad del Planeta. Por ello también hice que esta última entidad no tuviera porqué atraer a los demás objetos de la escena hacia sí mismo. Creé una entidad Gravedad y se la inserté a todas las entidades pertinentes. Con ello cambié el diagrama UML e inserté un atributo de Gravedad en las entidades que se veían atraídas, así como un atributo Munición en el Personaje:

7.3 Implementación

Para ajustar la cámara, la coloqué en proyección ortográfica para que apuntara a las coordenadas [0,0,0]. En estas coordenadas situé un plano con el fondo oscuro y otro plano del Planeta donde transcurre toda la acción en escena. También moví la luz de la escena y añadí al proyecto todos los Materiales para las diferentes apariencias del Planeta.

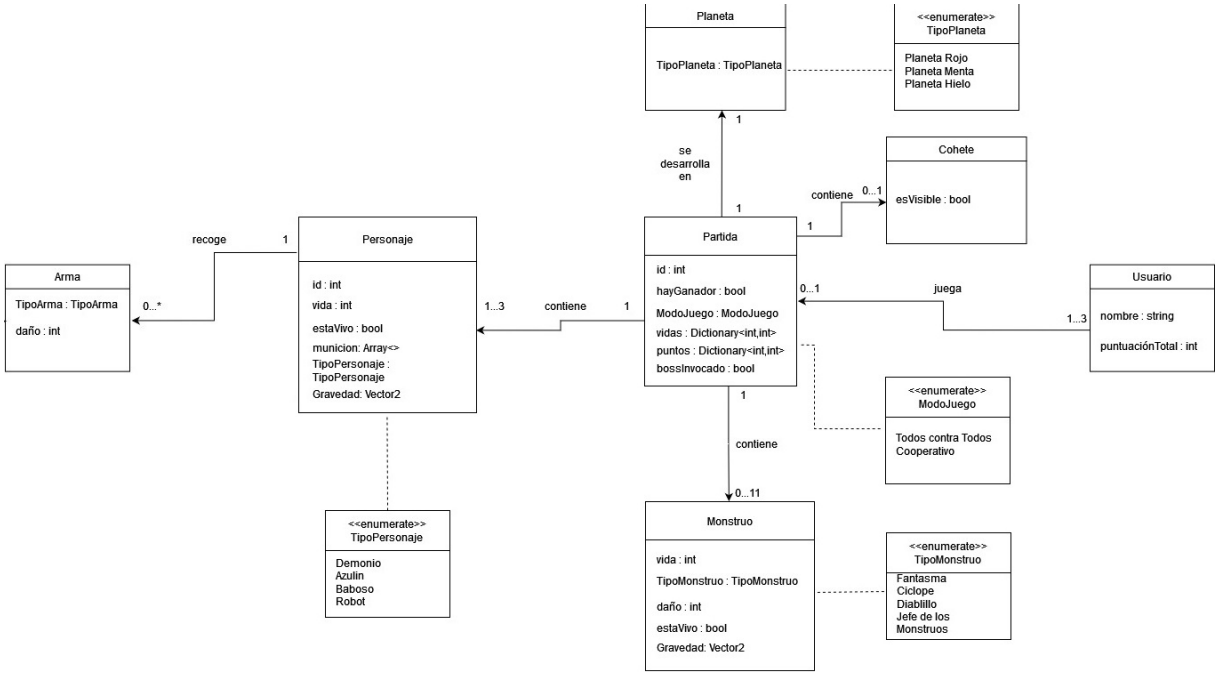

Figura 7.4: Diagrama UML de diseño: esquema final

A la hora de crear el resto de entidades apenas tuve problemas. Donde me surgió un imprevisto era a la hora de colocar el sistema de gravedad en la entidad Personaje. Cada vez que ejecutaba el juego veía como los Personajes atravesaban el Planeta a pesar de compartir un atributo *RigidBody2D* y un *Collider*. Más tarde me di cuenta de que el problema venía por la opción *IsTrigger* de su *Collider* por lo que al desactivarla en cualquier entidad no daba esos problemas.

Para explicar la gravedad voy a adjuntar unas líneas de código relevantes que ayudan a ver cómo actúan los vectores de atracción y rotación entre cualquier entidad y el Planeta.:

```
void applyGravity() {
    Vector2 direction = (Vector3.zero -
    gameObject.transform.position).normalized;
    float distance = Vector2.Distance(Vector2.zero,
    gameObject.transform.position);

    float force = gravity / (distance * distance);

    rb.AddForce(direction * force);
```

```
}

void applyRotation() {
    float lag = 90f;
    Vector3 planetCenter = Vector3.zero;
    Vector3 MonsterPosition = transform.position;

    float angle = Mathf.Atan2(MonsterPosition.y - planetCenter.y,
    MonsterPosition.x - planetCenter.x) * Mathf.Rad2Deg - lag;
    transform.rotation = Quaternion.Euler(0f, 0f, angle);
}
```

En la primera función vemos que se calcula un vector direccional y la distancia entre el centro del Planeta y la posición de la entidad. Para aplicar la fuerza, se accede al componente *RigidBody2D* de la entidad y se le aplica un cociente entre una variable y la distancia al cuadrado.

En cuanto a la rotación del objeto es algo más costoso de entender. Partimos de que existe un desfase de 90 grados para que todos los cuerpos estén alineados en la parte inferior de su aspecto. Lo que se hace es aplicar una función para calcular el arco tangente obtenido en radianes y posteriormente su equivalente en grados.

7.4 Desviación temporal

Como he comentado en capítulos anteriores, he gestionado el tiempo con *Clockify*, una herramienta web para medir la duración de las tareas y clasificarlas según el tipo. A medida que iba desarrollando activaba el cronómetro y lo paraba cada vez que hacía una pausa o bien completaba una tarea.

En la siguiente lista se encuentra la duración total por conjunto de características segmentadas del actual apartado:

1. **El Planeta:** 5,32 horas.

2. **Los Personajes:** 15,8 horas.

3. **Los Monstruos:** 3,97 horas.

Después de ver la duración real de cada característica del juego, toca analizar el coeficiente de desviación temporal de cada una:

1. **El Planeta:** 5,32 / 6 = 0,89
2. **Los Personajes:** 15,8 / 7 = 2,26
3. **Los Monstruos:** 5,97 / 4 = 1,49

Desviación total de la iteración: 27,09 / 17 = 1,6

A groso modo, esto quiere decir que he empleado un 60 % del tiempo total estimado para esta iteración. Esto es comprensible ya que tuve que volver a refrescar muchos conceptos de desarrollo en *Unity* y preparar el proyecto para todo lo que vendría a continuación.

Iteración 2

En esta iteración desarrollamos el proyecto con características fundamentales en el resultado del proyecto. El código empezará a verse más denso y aparecerán más problemas.

8.1 Características a desarrollar

4. Los proyectiles y Armas - 14 horas

1. **Creación de la entidad Misil** (4 horas): esta entidad será disparada tanto por un Personaje como por un Monstruo y quitará puntos de salud sobre el Personaje o Monstruo sobre el que impacte.
2. **Creación de la entidad SableLaser** (4 horas): entidad que se basa en realizar una animación cuya área de efecto resta puntos de salud de Personajes o Monstruos que se encuentren en el radio de ataque.
3. **Creación de la entidad RayoLaser** (3 horas): esta entidad es similar a la entidad Misil solo que atraviesa la escena en línea recta y hace un daño mayor a cualquier Personaje o Monstruo.
4. **Creación de la entidad Mina** (3 horas): esta entidad se basa en dejar plantada una especie de bomba cuya activación ocurre cuando algún Monstruo pasa sobre ella.

5. El sistema de trazado y trayectorias - 9 horas

1. **Rectificar la trayectoria y físicas del Planeta** (4 horas): cada entidad disparada de tipo Misil tendrá una trayectoria parabólica que dependerá del ángulo con la que salga disparada, haciendo daño a cualquiera de las entidades que reciban el impacto.
2. **Dibujar la trayectoria al disparar** (5 horas): dibuja algunas líneas en trazo discontinuo sobre la trayectoria que va a tener el Personaje cuando esté en Modo Disparo y poder intuir la trayectoria del disparo.

6. Un inventario de armas para cada personaje - 3 horas

1. **Atributo por cada Arma** (2 horas): el Personaje cada vez que quiera disparar, debe elegir entre qué armas poder usar. Por tanto, necesita un atributo para cada arma que pueda utilizar.
2. **Habilidad para recoger la Munición** (1 hora): cada vez que un Personaje pasa sobre un objeto en la escena de Munición, puede aumentar la Munición que puede usar del Arma indicada.

8.2 Diseño

A la hora de crear los proyectiles y armas, todas estas entidades siguen el mismo esquema. Primero se crea el fichero con los atributos necesarios, como por ejemplo si es un proyectil que se lanza con una trayectoria, se deja caer o simplemente es una animación. Luego se ajusta su máquina de estados y apariencia en escena y por último se enlaza con las entidades Personaje y Monstruo para que la usen cuando quieran usar el arma deseada.

Memorando técnico 0002	
Asunto	Animación de explosión.
Resumen	Mantenerlo como una entidad sin dependencias en escena.
Factores causantes	La máquina de estado no funciona bien si la entidad padre está desactivada.
Solución	Se sitúa de manera perpetua en escena y se mueve en función del lugar de detonación del Misil en el Planeta.
Motivación	Se quiere tener la entidad Explosión controlada y cuya animación no se altere a lo largo de una partida.
Cuestiones abiertas	Se podría investigar sobre una manera de poner un efecto de explosión sin necesidad de crear una entidad con animaciones usando el sistema de efectos dentro de *Unity*.
Alternativas	Se podría activar la animación una vez la entidad cambie de ubicación si tenemos de referencia que siempre la situamos fuera de la escena en las mismas coordenadas.

Cuadro 8.1: Memorando técnico 0002

Al empezar la animación Explosión e insertarla en Misil, pensaba dejar el objeto Explosión heredado dentro de Misil. Sin embargo, no podía ajustarlo ya que activar Misil implicaba activar la animación antes de tiempo, por lo que el objeto está activo en escena perpetuamente en un espacio alejado y cada vez que Misil se desactiva, este se transporta momentánemente para simular el fenómeno de la explosión.

Para el sistema de trazado con tener una referencia visual simple es suficiente. Como veremos más tarde, esta solo se activa cuando un Personaje está en Modo Disparo y consta de una repartición espacial en línea recta de puntos que contrastan con el entorno.

8.3 Implementación

Tras empezar a desarrollar la funcionalidad de creación de la entidad Misil, hubiese sido buena idea haber creado antes la entidad Mina ya que desde Misil debe emitir daño a esta entidad. Por tanto, no ha sido una buena idea empezar una tarea donde necesito de otra funcionalidad terminada similar a ella para crearla y asegurarme que ambas estén completas.

En la creación de la entidad RayoLaser he intentado que se pintaran de manera adecuada las animaciones y que los retrasos frente a lo previsto en el desarrollo se deben a que he tenido problemas a la hora de ajustar la entidad RayoLaser cuando estaba siendo activada y disparada en la escena. Después de estas dificultades, decidí empezar de cero a cosechar una idea más simple pero efectiva con el concepto de RayoLaser que tenía inicialmente.

Memorando técnico 0003	
Asunto	Rediseño de entidad Rayo Laser.
Resumen	Pasar de animación compleja a proyectil móvil.
Factores causantes	El proyectil debe ser móvil, activarse cuando el Personaje pueda hacerlo sin recibir daño.
Solución	En lugar de crear una entidad que aumente su alcance por el tamaño de las animaciones, se vuelve un proyectil móvil que recorre todo el mapa.
Motivación	Se le da un aspecto más intuituvo como proyectil en lugar de emular la emisión de un haz de luz que puede causar errores y desplazamientos no deseados.
Cuestiones abiertas	Se podría crear empleando más tiempo en investigar cómo implementarlo sin que afecte a varios aspectos del juego.
Alternativas	Usar mejores animaciones, una máquina de estado que las ejecute de manera muy precisa y perfilar su alcance y posición entorno al Personaje.

Cuadro 8.2: Memorando técnico 0003

Para rediseñar este proyectil, cambié la expansión de un objeto que iba cambiando su tamaño mediante una entidad *Animator* de *Unity* por uno más pequeño que no cambia de forma y se mueve a lo largo del mapa en una misma dirección. La idea

inicial se basaba en una especie de esfera inmóvil donde elogándola con animaciones podía conseguir un efecto de disparo.

Esta nueva idea me resultó más fácil de realizar ya que tuve que repetir el proceso que hice cuando diseñé la entidad Misil. La unica dificultad radica en diferenciar entre el uso de *Colliders* y *Triggers* dentro de las físicas de *Unity* y mover la entidad a través de su atributo *Rigidbody2D*.

Para la entidad Mina apenas hubo problemas ya que en un principio me basaba en la creación de las armas anteriores. Inicialmente la idea era cambiar los atributos de la entidad para acercarlo y alejarlo de la partida dependiendo de si estaba siendo usada en la partida. En lugar de desactivar la entidad, se pone su ubicación en escena alejado del campo de visión y se cambia su atributo *Rigidbody2D* para que no le afecte la gravedad temporalmente.

Luego de haber implementado la entidad Misil, se ha rectificado la trayectoria que tenía cuando se movía en escena y se ha modificado su componente de Gravedad hacia el Planeta. Cuando esta entidad sale de los límites visuales en partida se desactiva así como si hubiera impactado contra el Planeta.

Donde he conseguido plantear una solución bastante ingeniosa y simple es a la hora de implementar el trazado de la trayectoria en el Modo Disparo de los Personajes. Cuando se activa este modo, se generan 5 puntos blancos equidistantes entre la posición del jugador y el cursor. Esto hace que en disparos de Rayo Láser se precise mejor la trayectoria y en disparos de Misil se intuya la trayectoria que puede seguir, haciendolo más difícil al jugador pues se tienen balas infinitas de este arma.

Luego de bastantes elementos implementados, experimenté algunos fallos imprevistos cada vez que probaba el juego en alguna partida. Para corregir fallos que no contemplé en la planificación como el cambio de trayectoria de las armas si el personaje se movía. Para ello separé las entidades en conjuntos fijos en la escena para que no estuvieran anclados a la posición del Personaje. Además, organicé el código para que cada entidad invocada lo hiciese en un fichero independiente y no mezclar código que no tuviese nada que ver.

A partir de aquí decidí que la Munición, Personajes, Monstruos así como cualquier otro entidad móvil dentro de la escena fuese movido a las coordenadas precisas dentro o fuera de la vista del jugador, así como manipular en este proceso sus componente de físicas para que en escena siempre estuviese activo a la hora de gestionar cambios §8.3. Es decir, cada entidad estaba instanciada en escena y solamente necesitaba modificar

su aparición y físicas en escena. Un ejemplo de esto también se puede apreciar en la actualización en cada turno de la inercia de cada entidad en escena, para que no se mueva perpetuamente de manera arbitraria.

Análisis de valor aportado 0001	
Propuesta	Manejo de entidades en escena con *Object Pooling*
Valor	Mejor rendimiento dentro del juego con mejores prácticas usadas por la industria así como simplificar el número de entidades en escena.
Coste	Generar una mayor cantidad y complejidad en la legibilidad del código.
Opciones	Usar métodos para crear y destruir entidades durante la ejecución del juego.
Riesgos	Posible número elevado de entidades en escena cuando se carga el juego pues se procesan antes de ser creadas a medida que avanza un juego.
Deuda técnica	Tener que replicar esta práctica a medida que se extiende el proyecto en contenido

Cuadro 8.3: Análisis de valor aportado 0001

Por último, indicar el cambio sobre la entidad Mina. Si un Personaje ha colocado 2 minas en el Planeta, se destruirá la que se haya colocado antes y se pondrá en el sitio indicado.

8.4 Desviación temporal

En la siguiente lista se encuentra la duración total por conjunto de características segmentadas del actual apartado:

1. **Los Proyectiles y Armas:** 34,07 horas.
2. **El sistema de trazado y trayectorias:** 12,07 horas.
3. **Un inventario de armas para cada personaje:** 14,17 horas.

Después de ver la duración real de cada característica del juego, toca analizar el coeficiente de desviación temporal de cada una:

1. **Los Proyectiles y Armas:** 38,57 / 14 = 2,76
2. **El sistema de trazado y trayectorias:** 14,57 / 9 = 1,62
3. **Un inventario de armas para cada personaje:** 5,17 / 3 = 1,72

Desviación total de la iteración: 58,31 / 26 = 2,24

En esta iteración me desvié empleando más del doble del tiempo sobre todo haciendo las entidades de las armas. Tuve dificultades a la hora de programar los disparos pues las entidades de los proyectiles no tenían mucha complicación. Esto se debe a la dificultad de trabajar con *RigidBody2D* usando varios componentes vectoriales diferentes como la Gravedad y los parámetros de velocidad de los proyectiles al ser disparados.

ITERACIÓN 3

Estas características son las últimas antes de empezar con las tareas de interfaces de juego, por lo que a partir de aquí se tendrá casi toda la jugabilidad dentro de las partidas.

9.1 Características a desarrollar

7. El sistema de acción por turnos - 14 horas

1. **Inserción de lógica de turnos individual** (8 horas): aquí se gestionará en otro fichero las acciones y el estado de la entidad que pueda moverse en ese turno.

2. **Configuración inicial en GameManager** (6 horas): implementar la lógica de turnos con el trascurso de turno actual y el cambio de turno.

8. Configuración de la IA de Monstruos - 6 horas

1. **Subrutina de toma de acciones** (6 horas): se creará un conjunto de métodos para que los Monstruos puedan tomar acciones en función de las entidades Personaje que haya en escena o bien de manera aleatoria.

9. El Cohete - 5 horas

1. **Creación de la entidad Cohete** (2 horas): insertar un *PNG* de Cohete en la escena para posteriormente incluirlo en las partidas.

2. **Configuración para la aparición del Cohete** (3 horas): cuando en una partida los puntos de salud del Jefe de los Monstruos lleguen a 0 aparecerá el Cohete.

10. La gestión de partidas - 14 horas

1. **Aparición de Personajes, Monstruos y Munición** (5 horas): por el mapa, irán apareciendo Monstruos y Munición a medida que pasen los turnos.

2. **Insertar Puntos en las partidas** (3 horas): cuando un Personaje derrote a un Monstruo o gane una partida, el Jugador recibirá Puntos. Estos puntos quedarán en el perfil del Jugador.

3. **Insertar Vidas en las partidas** (2 horas): un Jugador tendrá 3 Vidas por partida. Si las pierde, la partida termina y obtendrá los Puntos que haya conseguido hasta ese punto de la partida. Cada vez que los Puntos de salud del Personaje llegan a 0, se resta el número de Vidas en 1 y ese Personaje vuelve a aparecer en el punto inicial donde apareció.

4. **Finalizar partidas** (2 horas): cuando un Personaje haya llegado al Cohete o bien haya agotado las Vidas que tenía, se terminará la partida.

5. **Creación de modos de juego** (7 horas): el Jugador podrá jugar en modo Cooperativo, controlando a 3 Personajes en total o bien en modo Todos contra Todos donde solo podrá controlar a uno y derrotar por sí mismo al Jefe de los Monstruos.

9.2 Diseño

Para el transcurso de la partida se va a crear un sistema de gestión de turnos. De esta manera, solo una sola entidad puede moverse libremente y tomar acciones a la vez durante un tiempo limitado y de manera ordenada. Esto quiere decir que ninguna entidad va a poder moverse dos veces por delante de una entidad que siga viva en escena ya que existe un orden dentro de cada partida a medida que se añaden o quiten entidades.

Este sistema de gestión de turnos primero va a aumentar en una unidad por cada turno transcurrido y aparte va a tener en cuenta la posición de hijos que tenga la entidad Planeta adheridos en escena. De esta manera, se aplica al conteo de turnos actual la operación modular del número de entidades en el Planeta.

Cada entidad tiene una variable booleana sobre si es su turno o no para ver si pueden actuar. Si tiene un valor de *True* esa entidad se puede mover ya que es su turno y se ve anulada si tiene el valor *False*. Pues a continuación del cálculo de los turnos se pone el valor *False* en cada entidad en el Planeta y se asigna *True* solo a la que debe moverse en ese turno. Transcurrido el tiempo restante, se vuelve a empezar todo desde el principio.

9.3 Implementación

La inserción de Turnos dentro del juego pensaba que me iba a costar más trabajo. Seguí un esquema de desarrollo simple, donde el fichero ejecuta de manera recurrente una serie de funciones para llevar la lógica de los turnos:

```
public void shiftController() {
    shiftTimer -= Time.deltaTime;
    secondsleft.text=shiftTimer.ToString("F1") + _secondsleft;
    if(shiftTimer <= 0f) {
        resetMobility();
        calculateTurn();
        checkSpawning();
        shiftCounter++;
        totalShifts++;
        shifts.text = _shifts + totalShifts;
        shiftCounter = shiftCounter % mobiles.Count;
        shiftTimer = GameManager.instance.shiftDuration;
    }
    getMobileInfo();
    detectPauseKeyCode();
    checkGameStatus();
}
```

Con *shiftTimer* tenemos un contador del tiempo restante del turno actual. Cuando este llega a 0 empieza la asignación del siguiente turno a otra entidad. Primero, se ejecuta una función para que cualquier entidad con una inercia descontrolada se frene. En la función siguiente se asigna el siguiente turno en función del contador y se verifica además de un turno a otro el jugador a perdido la partida. El resto de métodos son simplemente para aumentar el conteo de turnos y hacer aparecer más Monstruos o Munición.

También se puede apreciar código relativo al *HUD* y otros menús pues es el método sacado de la versión final del proyecto.

Con respecto a la IA de los Monstruos he decidido que su movimiento y dirección fuesen aleatorias ya que así se ofrece una mayor lectura de lo que acontece por turnos en cada partida. Lo único que varía aquí es el Jefe de los Monstruos pues casi siempre dispara hacia un Personaje aleatorio y a medida que hay más Personajes vivos en la partida, la tasa de fallo disminuye.

Los modos de juego los integré también de manera sencilla, haciendo que la puntuación conseguida por partida se pudiese almacenar siempre en el mismo vector. Al cambiar el modo de juego, se usaba el vector de cierta manera pues la idea es almacenar toda la puntuación en el Modo Cooperativo en la primera posición o repartir la

puntuación a razón de una posición para cada Personaje en partida.

A la hora de implementar los demás aspectos del juego dentro del juego, se corrigieron bastante errores de jugabilidad y por ello me retrasé a la hora de subir la funcionalidad completa. Entre estos errores diferenciamos la asignación de Personajes al Planeta cuando están vivos o ajustar las animaciones de Sable Láser.

Análisis de valor aportado 0003	
Propuesta	Asignación de puntuación por modo de juego
Valor	Se manipula el conjunto de valores de una lista en función de un parámetro booleano para que sea directa la lectura de información en escenarios posteriores.
Coste	Añadir una evaluación en torno a una variable para sumar en el primer elemento de la lista o repartirlo en torno a tres.
Opciones	Crear un diccionario con el número de puntuación de cada Personaje y calcular la puntuación al terminar la partida.
Riesgos	Mayor complejidad computacional y peor legibilidad de código.
Deuda técnica	Casi inexistente al tratar una lista en lugar de variables separadas.

Cuadro 9.1: Análisis de valor aportado 0003

Concretamente a la hora de insertar la división de puntos en Modo Cooperativo o Todos contra Todos, lo que cambia en el código es los lugares en los que se almacena la puntuación y qué Personajes se mueven automáticamente.

9.4 Desviación temporal

En la siguiente lista se encuentra la duración total por conjunto de características segmentadas del actual apartado:

1. **El sistema de acción por turnos:** 12,9 horas.
2. **Configuración de la IA de Monstruos:** 7,32 horas.
3. **El Cohete:** 6,38 horas.

4. **La gestión de partidas:** 17,18 horas.

Después de ver la duración real de cada característica del juego, toca analizar el coeficiente de desviación temporal de cada una:

1. **El sistema de acción por turnos:** 12,9 / 14 = 0,92
2. **Configuración de la IA de Monstruos:** 7,32 / 6 = 1,22
3. **El Cohete:** 6,38 / 5 = 1,28
4. **La gestión de partidas:** 17,18 / 14 = 1,23

Desviación total de la iteración: 43,78 / 39 = 1,12

En esta iteración me he desviado ligeramente ya que supuse que el sistema de turnos iba a costar algo más de tiempo. De todas formas la IA que he implementado tanto en Monstruos como en Personajes que no son controlados por el jugador actúa de manera bastante arbitraria para crear incertidumbre. Por ello el tiempo estimado ha sido menor del que realmente podría haber llevado si hubiese tenido en cuenta más aspectos.

Iteración 4

No todo lo restante en materia de desarrollo versa en esta iteración. Sin embargo ésta ocupa un gran número de horas al tener que desplegar todo el apartado de interfaces del proyecto.

10.1 Características a desarrollar

11. Menús de juego - 30 horas

1. **Inserción de HUD** (6 horas): el jugador podrá ver en todo momento información relevante sobre su personaje, como sus puntos, vidas restantes, munición restante y arma seleccionada.
2. **Menú de Inicio** (4 horas): se creará un Menú que aparecerá cuando el Jugador inicie el juego, que lo llevará a las pestañas de 'Jugar', 'Opciones' y 'Salir del Juego'.
3. **Pestaña de Opciones** (8 horas): aquí se podrá manipular el volumen de juego y desactivar las animaciones de impacto de Misil.
4. **Menú de Pausa y Menú de Fin de Partida** (4 horas): este primer menú permite pausar una partida que está en transcurso con las opciones 'Reanudar', 'Reiniciar partida' y 'Salir del juego'. El otro es una menú más simple, con una vista reducida que lleva directamente al menú principal, haciendo lo mismo que el botón 'Terminar partida' solo que de manera automática.
5. **Menú de Selección de Personajes** (5 horas): este menú permite elegir la apariencia del personaje que se va a utilizar durante la partida. Es un menú que se encuentra después de acceder a la pestaña de 'Jugar' y antes de cada partida.
6. **Menú de Selección de modo de juego** (3 horas): previamente a cada juego y después de elegir el Personaje a usar, el Jugador podrá elegir el modo de juego preferido: Cooperativo o Todos contra Todos

12. Una base de datos en la nube - 10 horas

1. **Configuración de un SGBD con Firebase** (10 horas): crear una base de datos en Firebase con los datos de cada Jugador: Usuario, contraseña, Puntos y objetos desbloqueados.

13. Un formulario de inicio de sesión y registro - 12 horas

1. **Creación de gestión de credenciales** (3 horas): crearé un fichero para evaluar los casos de inicio de sesión y registrar un nuevo usuario.

2. **Formulario de inicio y registrar usuario** (3 horas): realizaré un formulario simple de inicio de sesión y registro de usuarios.

3. **Enlace de SGBD con el formulario** (6 horas): usaré la base de datos en Firebase para validar los usuarios que se puedan registrar e iniciar sesión con el formulario inicial.

14. Sistema de puntuación y Tienda - 23 horas

1. **Contador de puntos por Jugador** (4 horas): todos los Puntos conseguidos en una partida terminada irán al contador de Puntos de cada Jugador.

2. **Incluir pestaña de Tienda en Menú de Inicio** (1 hora): el Jugador podrá acceder a una Tienda desde el Menú de Inicio del juego

3. **Creación de Tienda** (12 horas): cada jugador podrá acceder a una tienda donde podrá obtener Personajes jugables.

4. **Ajuste de partidas por objetos canjeados en tienda** (6 horas): en función de los Personajes desbloqueadas por el jugador, en las partidas se podrá elegir la apariencia del Personaje y del Planeta.

10.2 Diseño

El esquema de desarrollo en esta iteración consta del aprendizaje y familiarización de la herramienta *UI Toolkit* empezando por los menús más simples y necesarios. Más tarde, nos adentraremos en la creación de las interfaces más complejas teniendo en cuenta su diseño visual y la interconexión con los datos de un fichero JSON.

Ésta es una herramienta novedosa y que sustituye a la forma anterior de crear el canvas o espacio con toda la información pertinente a los datos de la partida.

Primero se tiene que crear un objeto en escena, asignarle un atributo *UI Document* para que cargue un fichero visual que podemos editar dentro de *Unity*. Una vez que hayamos editado y asociado este fichero tendremos que asignar otro que contenga la lógica para presentar los datos una vez queramos situar el menú correspondiente al jugador.

Cada fichero visual puede compartir la misma configuración CSS para estilizar la fuente u otros elementos visuales de manera común. Así, se consigue tener un conjunto

de interfaces de una misma temática pudiendo reusar y asignar estilos con el uso de etiquetas.

Dentro del fichero con el código para conectar la interfaz con los datos, se tiene una estructura parecida para cada menú. Primero se instancia una variable con el documento del menú y se asignan en distintas varibles los elementos con los que se quiere representar la información.

Más tarde, se registra el cambio para los elementos interactivos como botones, menús desplegables o barras de selección. Aquí es donde a cada elemento se le asigna una función a ejecutar que definirá el comportamiento de la interacción del jugador con el elemento accedido.

El objetivo de almacenar datos por jugador se pide para que el uso de una Tienda y el conseguir Puntos tenga un sentido. Además se crea la sensación de recompensar al jugador. Por ello, para este prototipo con tener un repositorio local de información se simplifica mucho el proceso dentro del proyecto. Concretamente se guardan las credenciales y el número de puntos con los objetos comprados con estos.

Para los aspectos de la Tienda se busca que al tener puntos suficientes y querer comprar un elemento visual, el jugador lo tenga para futuras partidas. Así, el juego detecta los elementos ya comprados y restringe al jugador de poder comprarlos de nuevo. Si no tiene Puntos suficientes se muestra un mensaje de error.

10.3 Implementación

A partir de aquí, casi toda la implementación de código es relativa a los menús que decidí añadir, es decir que no añadí más funcionalidades a aspectos de cada partida en el juego. Al no estar familiarizado con la nueva herramienta de edición de interfaz de usuario de *Unity*, tuve que investigar bastante acerca del tema y por ello invertí mucho más tiempo del previsto.

A medida que iba haciendo los menús he decido reutilizar recursos como unificar el Menú de Fin de Partida tanto si se gana una partida como si se pierde o usar el mismo icono de botón para todos los elementos del juego.

Otro elemento que decidí cambiar cuando me topé con él fue el guardado de datos del jugador dentro del videojuego. Y es que a la hora de probar ciertas funcionalidades de manera continua prefería guardar la información usando un archivo con forma-

to JSON de manera local en lugar de usar un sistema de guardado en la nube como Firebase.

Evaluando las ventajas y desventajas, la creación de un servidor de Firebase no perduraría en el tiempo y además, sería más costoso de implementar y modificar datos. En el punto en el que estaba dentro del desarrollo, alargar las horas de desarrollo por medio de esa tarea no sería objeto de estudio dentro del presente trabajo.

Las ventajas que ofrece el uso de un archivo con extensión JSON es la rapidez a la hora de modificar la base de datos del juego, así como la facilidad a la hora de consultar y probar información y métodos de prueba.

Memorando técnico 0003	
Asunto	Guardado de credenciales.
Resumen	Uso de archivo extensión JSON.
Factores causantes	Se debe poder crear y acceder a una cuenta de preferencia por el usuario.
Solución	Uso de un archivo de extensión JSON para poder almacenar esta información, así de no depender en este caso de una conexión a Internet.
Motivación	Al ser un proyecto simple donde no se aprecia una faceta de conexión a Internet se opta por una alternativa más simple para resolver el problema.
Cuestiones abiertas	No existe ningún tipo de seguridad para asegurar que no se modifican los datos de cada jugador.
Alternativas	Usar un sistema de gestión de base de datos en la nube para asegurar las credenciales en cualquier equipo y no solo de manera local. Al ser un proyecto que no tiene de primera mano publicarlo en el mercado, la solución más eficaz es usar un archivo local.

Cuadro 10.1: Memorando técnico 0003

10.4 Desviación temporal

En la siguiente lista se encuentra la duración total por conjunto de características segmentadas del actual apartado:

1. **Menús de juego:** 13,14 horas.
2. **Una base de datos en la nube:** 11,67 horas.
3. **Un formulario de inicio de sesión y registro:** 8,12 horas.
4. **Sistema de puntuación y Tienda:** 16,77 horas.

Después de ver la duración real de cada característica del juego, toca analizar el coeficiente de desviación temporal de cada una:

1. **Menús de juego:** 13,14 / 30 = 0,44
2. **Una base de datos en la nube:** 11,67 / 10 = 1,17
3. **Un formulario de inicio de sesión y registro:** 8,12 / 12 = 0,68
4. **Sistema de puntuación y Tienda:** 16,77 / 23 = 0,73

Desviación total de la iteración: 49,7 / 65 = 0,76

La dificultad de esta iteración radica en saber gestionar bien cómo administrar los datos de usuario. Al haber optado por crear un JSON en lugar de usar algún sistema para almacenar los datos en la nube, pude recortar bastante tiempo del estimado. No obstante, la parte de diseñar los menús no ha sido incluida ya que no ocupa tiempo de desarrollo de código. Sin embargo apenas subiría la desviación ya que no son más que **2,16** horas extra.

Iteración 5

Con este iteración termina el desarrollo del proyecto. Además de ver los últimos detalles, veremos el entorno de pruebas en el que voy a acotar el proyecto y la puesta en marcha del despliegue.

11.1 Características a desarrollar

15. Música ambiental y sonidos - 13 horas

1. **Insertar música de ambiente y música de partida** (5 horas): dentro del juego sonará una música de ambiente y durante el transcurso de la partida sonará otra melodía diferente.

2. **Insertar efectos de sonidos** (6 horas): tanto los Monstruos durante las partidas, como el Jefe de los Monstruos, ganar una partida y el disparo e impacto de los proyectiles tendrán sonidos.

3. **Enlazar pestaña Opciones con el volumen de juego** (2 horas): se enlazará la modificación de música y sonido con la pestaña del menú Opciones.

16. Animación de efectos - 7 horas

1. **Insertar animación de proyectiles** (3 horas): tanto la entidad Misil como la entidad Mina tendrán un efecto de explosión cuando se detonen.

2. **Enlazar pestaña Opciones con los efectos** (2 horas): se enlazará la aparición de efectos con la pestaña del menú Opciones.

3. **Traza de estela a Misil** (2 horas): para que la entidad Misil sea más visible debido a la fugacidad de su duración en escena, se decide implementar un trazo en la trayectoria de su lanzamiento.

11.2 Diseño

Para este videojuego vamos a reproducir una pista de audio contínuamente así como sonidos cuando el jugador siga interactuando con los elementos visuales de las interfaces. [[1]]

Durante todo el proyecto he estado usando tanto modelos en *Pixel Art* para Personajes, Munición, etc. Sin embargo a veces he tenido que modificar o crear algunos simples desde cero pues en Internet no encontré los que necesitaba. Por ello tomé la decisión de a medida que empezaba el desarrollo de una entidad, tener los modelos en *Pixel Art* previamente al desarrollo. §11.1

Análisis de valor aportado 0002	
Propuesta	Creación de elementos simples con *Pixel Art*
Valor	Se consiguen diseños de entidades que se traduciría en tiempo y/o dinero extra para conseguir soluciones válidas y que se adapten a la escala de tamaño del videojuego. Además se evita el problema de derechos de autor en ciertos elementos que están publicados.
Coste	Aprender habilidades distintas aunque no computables para el cómputo del proyecto.
Opciones	Usar elementos de diseño gráfico de pago.
Riesgos	Gasto extra de tiempo y esfuerzo para el desarrollador, así como una apariencia menos profesional a nivel de detalles.
Deuda técnica	Inexistente ya que no influye en el código ni ralentiza procesos.

Cuadro 11.1: Análisis de valor aportado 0002

11.3 Implementación

A la hora de implementar los elementos restantes en la última iteración, tuve en cuenta todo lo que adelanté añadiéndolo en funcionalidades anteriores. Por ello solo tuve que cambiar algunos parámetros, conectar ciertas funcionalidades y darle un acabado simple para que el producto ya si fuese algo jugable y disfrutable.

Lo único que estaba fuera de mis planes era la adición de un elemento visual a la entidad Misil. Esta decisión la tomé pocos días después de empezar dicha iteración pues el proyectil no estaba mucho tiempo en pantalla al ser lanzado. Por ello añadirle una estela a su paso era una buena opción para contemplar algo tan dinámico y que no constrastaba en pantalla por brillo y colores.

Memorando técnico 0004	
Asunto	Visibilidad de entidad Misil.
Resumen	Poder verlo mejor en escena.
Factores causantes	Debido al poco contraste de colores, se dificulta la visualización de la entidad en escena.
Solución	Se considera que añadiendo una estela de color blanco que resalte sobre lo demás y trazando la trayectoria recorrida por este en tiempo real tras su proyección se puede visualizar mejor.
Motivación	Se propone para que a la hora de realizar una prueba gráfica del juego existe una mayor compresión del escenario.
Alternativas	Se pueden cambiar los colores del Misil o su tamaño, aunque eso implique volver a rediseñar y alejarnos de la motivación inicial del problema.

Cuadro 11.2: Memorando técnico 0004

11.4 Pruebas

El conjunto de pruebas unitarias se componen de un total de 4 pruebas:

1. **checkMovementFunction**: en esta función se comprueba el vector de movimiento que se calcula cuando es el turno de un Enemigo. Para ello se obtiene la dirección del eje X que se calcula en otra función y luego se compara con el vector resultante.

2. **checkLoosingHealthFunction**: comprobamos la situación en la que un Personaje obtiene todos sus puntos de salud para luego quitarle una cantidad aleatoria. A continuación, vemos si la función de quitarle puntos de salud le ha dejado en el número correspondiente.

3. **checkPlanetPointGenerationFunction**: esta función calcula un punto cualquiera de la superficie del Planeta y se comprueba si la distancia desde este al centro del Planeta es constante.

4. **checkGrabingMunitionFunction**: podríamos decir que esta es la prueba más elaborada ya que comprueba y clasifica el tipo de Munición generada. Luego se al-

macena en el inventario de un Personaje nuevo y se comprueba si se ha añadido el tipo de Munición en su cantidad correspondiente.

11.5 Despliegue

A la hora hora de ejecutar el despliegue, tuve que cambiar la dirección del archivo JSON para leer y escribir información ya que el editor de *Unity* coge los datos de una ruta de datos propia del directorio del proyecto. Por el contrario a la hora de ejecutar el juego, se obedece a otra ruta del ordenador que en el caso de *Windows* crea un directorio en la carpeta %appdata%. Esta es la ruta persistente en la ejecución del juego como ejecutable y no como proyecto de *Unity*, por ello tuve que subir los cambios luego de haber hecho todo el juego.

Por otro lado, existe un fallo en la versión de *Unity* en la que estoy desarrollando el juego a la hora de cargar las interfaces de usuario cuando el juego se exporta como un ejecutable. Es decir, dentro de *Unity* el juego funciona perfectamente pero cuando se exporta y se ejecuta, el juego no puede procesar los menús cuando ocurre un cambio de escena. Por ello, decidí no cambiar de versión y crear una versión reducida y ejecutable del juego sin los menús ni información durante las partidas.

11.6 Desviación temporal

En la siguiente lista se encuentra la duración total por conjunto de características segmentadas del apartado y el fin del desarrollo:

1. **Música ambiental y sonidos:** 5,15 horas.
2. **Animación de efectos:** 5,6 horas.

Después de ver la duración real de cada característica del juego, toca analizar el coeficiente de desviación temporal de cada una:

1. **Música ambiental y sonidos:** 5,15 / 13 = 0,4
2. **Animación de efectos:** 5,6 / 5 = 0,72

Desviación total de la iteración: 10,75 / 18 = 0,6

Al haber implementado la mayoría de elementos relacionados a estos puntos en iteraciones anteriores, solo tenía que terminar de formalizar algunas tareas y completarlas para que la experiencia del proyecto fuera completa. Por otro lado, la implementación de la estela de disparo en Misil ha sido un añadido de última hora para que se pudiese visualizar bien el proyectil.

11.6.1 Desviación temporal total

Vamos a echarle un vistazo al diagrama generado por *Clockify* desde el inicio del proyecto hasta el desarrollo de la última funcionalidad

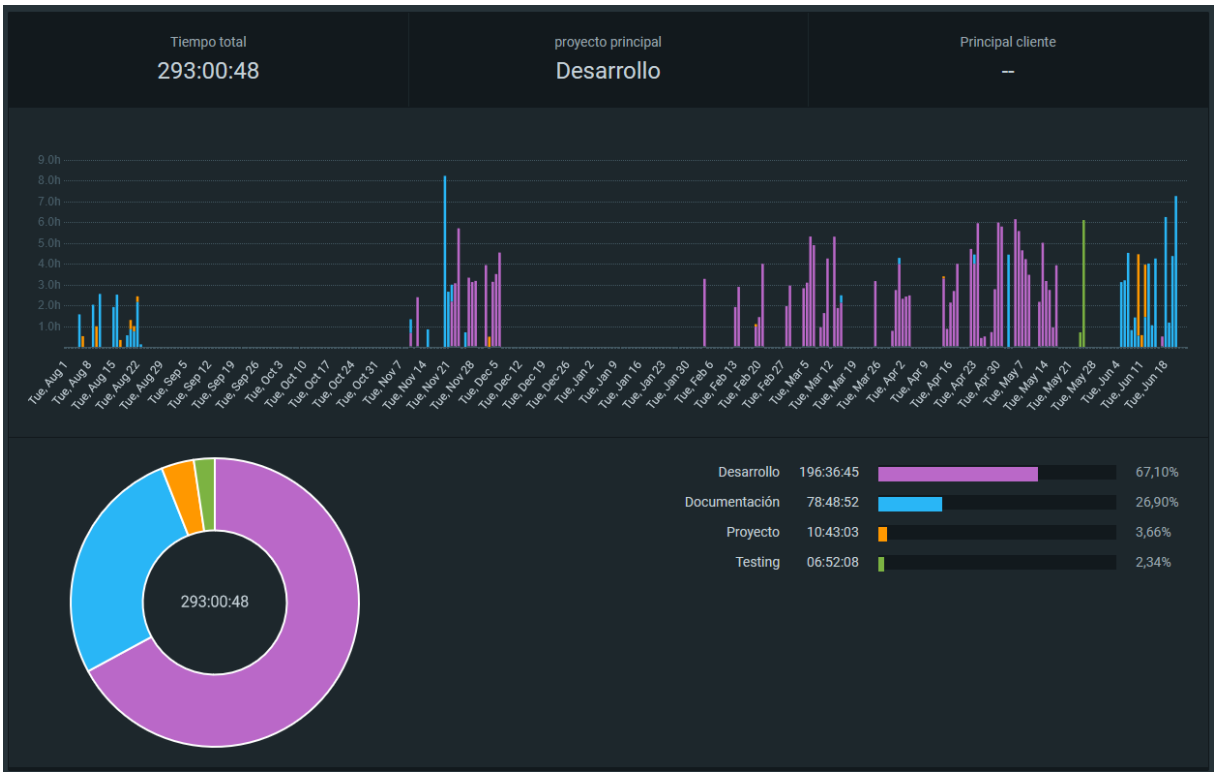

Figura 11.1: Diagrama temporal en Clockify

Desviación total del proyecto: 189,09 / 165 = 1,15

Esto quiere decir que he invertido un **15%** más del tiempo total estimado. Existen características como el apartado de *Los Personajes* en los que he invertido más de 2 veces el tiempo estimado en la planificación. No obstante, algunas características han ocupado en torno al 40% del tiempo total estimado, lo cual es algo menos de la mitad.

Parte IV

Cierre del proyecto

Conclusiones

Vamos a contar los aspectos más importantes de este proyecto con la experiencia que he tenido a la hora de desarrollarlo. Contaremos tanto los aspectos favorables como los desfavorables.

12.1 Informe post-mortem

El presente informe post-mortem se centra en el análisis de los eventos que rodearon el final de mi TFG. A continuación, se presentan los problemas y obstáculos que se enfrentaron, las acciones correctivas propuestas y los aprendizajes y lecciones aprendidas.

12.1.1 Lo que ha ido bien

- Tener un conocimiento previo de la tecnología de *Unity*, así como experiencia para trabajar con la documentación y lenguaje de programación.
- Trabajar sobre una idea predefinida y con un valor único en el mercado pues hacía que la motivación por el proyecto fuera mayor a tener que replicar algo en auge.
- Gestión del tiempo de manera exacta con *Clockify* ya que indicaba en cada momento cuándo empezaba a trabajar y cuándo paraba.
- A pesar de ser solo un único desarrollador, no he tenido que resolver conflictos a la hora de programar ni juntar ramas del repositorio pues todo lo ejecutaba desde mi equipo y trabajo desde la rama principal.
- Tener un archivo de texto personal en el que iba anotando los elementos que tenía que cambiar, un apartado de seguimiento y control sobre cada tarea para acelerar el proceso de documentarla y aspectos que iba asimilando en las tutorías y revisiones, así como líneas de código o comentarios que iba haciendo y me sirviesen más tarde.
- Usar herramientas de búsqueda basadas en IA para tener una visión adicional de como enfocar ciertos problemas, además de dar referencias a otros espacios bibliográficos de donde saca la información, véase ChatGPT [[14]] y Perplexity [[6]].

12.1.2 Lo que ha ido mal

- Trabajar en el proyecto y no tenerlo como prioridad alta, por lo que el trabajo se hacía discontinuado y eso implicaba volver a repasar código o mis apuntes personales para situarme tiempo después de haberlo hecho.

- Tener que cambiar algunas implementaciones o diseños por falta de recursos sin derechos de autor o experiencia a la hora de elaborar elementos.
- Estimación innexacta de algunas funcionalidades a la hora de desarrollarlas por falta de experiencia tanto en el desarrollo como en el lenguaje propio de *Unity*.
- Trabajar sobre una versión de *Unity* que implementa una herramienta diferente para elaborar las interfaces de usuario.

12.1.3 Discusión

En general los aspectos negativos no me han entorpecido mucho en el desarrollo. Sin embargo, antes de empezar otro proyecto similar lo primero que haría sería revisar bien qué versión de *Unity* se amolda mejor al proyecto. Otro aspecto sería hacer un calendario sobre qué días son innamovibles para desarrollar. Y por último, algo que hubiese ahorrado aún más tiempo y mejorado el código es definir las pruebas unitarias antes de hacer el código.

12.2 Trabajos futuros

Para proyectos futuros me gustaría contar con especialistas en el sector que me aconsejen sobre cómo resolver ciertos problemas que involucren esta tecnología, así como formarme por mí mismo para no estar atascado en una tarea dada. En este caso, hacer un videojuego 2D simplifica gran parte del proceso tanto en desarrollo, pruebas de jugabilidad y diseño gráfico. No obstante si no tuviese herramientas para el apartado gráfico o conocimientos sobre cómo hacer modelos en 3D sería un problema.

Parte V

Apéndices

Guía del Proyecto

Este apartado contiene información extra acerca del proyecto sobre un guía del proyecto a modo de manual de usuario para el jugador.

A.1 Manual de Usuario

Este juego está ambientado en un planeta con gráficos pixelados en dos dimensiones. El objetivo es ir consiguiendo Puntos a medida que vamos eliminando Monstruos con nuestras armas en un sistema de combate por turnos. Con esos puntos vamos consiguiendo nuevos personajes y diferentes Planetas en los que jugar.

Cada partida empieza con la aparición de tres Personajes que son los que controlaremos. A medida que pasan los turnos, van apareciendo Monstruos y Munición de las armas. Una vez hayamos eliminado diez Monstruos, aparecerá el Jefe de los Monstruos el cuál debemos eliminar para poder huir del Planeta.

Al derrotarlo aparecerá el Cohete en el que tocándolo ya habremos ganado la partida. Tanto si perdemos todas las Vidas de los Personajes como si tocamos el Cohete la partida terminará con la puntuación que hayamos conseguido aunque perder implica no ganar tantos Puntos.

A.1.1 Pantalla de Inicio de Sesión

Figura A.1: Pantalla de Inicio de Sesión

Una vez empecemos a jugar tendremos que meter las credenciales necesarias. Pode-

mos en el momento poner el usuario y la contraseña que precisemos pues se guardará en el archivo JSON antes mencionado. Cada vez que compremos con Puntos algún elemento dentro de la Tienda se guardará en nuestra cuenta, por lo que la información persiste al abrir y cerrar el juego.

A.1.2 Menú Principal

Figura A.2: Pantalla de Menú Principal

Una vez hayamos introducido las credenciales estaremos en el Menú Principal, donde podremos acceder a una partida desde el botón "Jugar", cambiar la configuración del juego en la pestaña .ºpciones.º salir del juego en la pestaña "Salir". También, podemos cambiar las credenciales en la flecha situada en la esquina inferior izquiera o acceder a la Tienda en la esquina inferior derecha.

A.1.3 Menú Opciones

Podemos desde esta pestaña cambiar el volumen de la música del juego, los efectos de sonido así como seleccionar la dificultad y la activación de efectos visuales en partida.

Figura A.3: Pantalla de Menú Opciones

A.1.4 Menú Tienda

Una vez en el menú de Tienda podremos ver la puntuación que tengamos arriba a la derecha. Si esta es mayor que el precio de alguno de los cosméticos para Personajes o Planetas, podremos gastar nuestros Puntos y usarlos en partidas futuras.

A.1.5 Menú Selección

Si desde el menú Principal hemos pulsado sobre "Jugar.estaremos en la pantalla previa al inicio de la partida. Una vez estemos aquí, podremos seleccionar la apariencia de los Personajes y Planetas. Además, justo antes de empezar la partida también podremos seleccionar el modo de juego.

A.1.6 Partida

Como podemos apreciar, este sería el aspecto final del juego. Tenemos a los Personajes en escena que podemos controlar así como un Monstruo en la parte inferior derecha del Planeta y Munición en la parte inferior izquierda. Con una flecha sobre la cabeza del Personaje sabemos el turno correspondiente para moverlo.

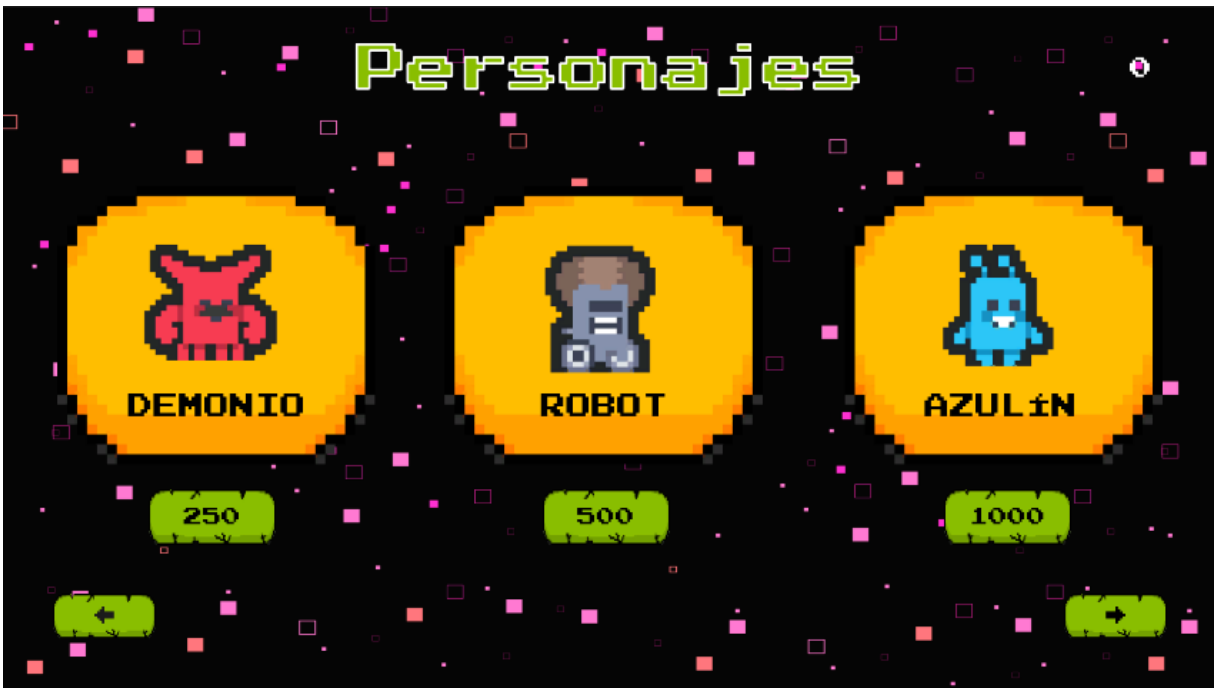

Figura A.4: Pantalla de Menú Tienda: Personajes

Figura A.5: Pantalla de Menú Tienda: Planetas

Figura A.6: Pantalla de Menú Selección

Figura A.7: Partida recién comenzada

Fijándonos ahora en los bordes de la pantalla tenemos la información de la entidad en su propio turno. En la parte superior de izquierda a derecha se sitúan la puntuación, número de Vidas, temporizador e información del sistema de turnos. En la parte inferior vemos la Munición, puntos de salud y arma seleccionada al activar el Modo de Disparo en ese mismo orden.

Figura A.8: Partida ganada

Si eliminamos al Jefe de los Monstruos y huimos en el Cohete nos saltaría esta pantalla de victoria.

Si por el contrario nos arrebatan todas las vidas, aunque conservemos la puntuación de igual manera nos saltará esta otra vista de derrota.

Figura A.9: Partida perdida

Bibliografía

[1] 8059346. Música y efectos de sonido. *Sitio Web*, 2024. Visitado desde el 27 de abril de 2024. (pages 26 y 58).

[2] Autores sin clasificar. *Technical Design Document and Game Design Document*. *Sitio Web*, 2023. Visitado desde el 11 de agosto de 2023. (page 12).

[3] BDragon1727. *Free Effect and Bullet 16x16*, en itch.io. *Sitio Web*, 2023. Visitado desde el 3 de diciembre de 2023. (page 31).

[4] P. Coad, E. Lefebvre, and J. D. Luca. *Java Modeling In Color With UML: Enterprise Components and Process*. *Prentice Hall International*, 1999. ISBN 0-13-011510-X. (page 12).

[5] codeman38. *Press Start 2P Font*. *Sitio Web*, 2024. Visitado desde el 28 de abril de 2024. (page 31).

[6] J. H. y. A. K. D. Yarats, A. Srinivas. *Perplexity AI,*. *Sitio Web*, 2023. Visitado desde el 6 de agosto de 2023. (page 66).

[7] Dawson Student. *Game Design Document*. *Sitio Web*, 2023. Visitado desde el 11 de agosto de 2023. (page 12).

[8] Dawson Student. *Technical Design Document*. *Sitio Web*, 2023. Visitado desde el 11 de agosto de 2023. (page 12).

[9] Deep-Fold. *Pixel Planet Generator*. *Sitio Web*, 2023. Visitado desde el 16 de agosto de 2023. (pages 26 y 31).

[10] Freepik. *Background pixel rain abstract*; fondo de pantalla cósmico. *Sitio Web*, 2023. Visitado desde el 10 de agosto de 2023. (page 26).

[11] Kenney. *Creature Mixer*, en itch.io. *Sitio Web*, 2023. Visitado desde el 10 de agosto de 2023. (page 26).

[12] D. Kvarfordt. *Pyxel Edit*. *Sitio Web*, 2023. Visitado desde el 3 de diciembre de 2023. (page 31).

[13] Meburningslime. *Free Online Art Community and Pixel Art Tool*, en itch.io. *Sitio Web*, 2023. Visitado desde el 3 de diciembre de 2023. (page 31).

[14] OpenAI. *ChatGPT*,. *Sitio Web*, 2023. Visitado desde el 6 de agosto de 2023. (page 66).

[15] Pixilart. *Free Online Art Community and Pixel Art Tool*. *Sitio Web*, 2023. Visitado desde el 3 de diciembre de 2023. (page 31).

[16] Project Management Institute. Guía del pmbok® (6ta ed.). *Newtown Square, PA: Project Management Institute*, 2017. (page 12).

[17] Statista. Distribución de los jugadores de videojuegos en españa en 2022, por edad y género. *Sitio Web*, 2023. Visitado desde el 12 de agosto de 2023. (page 4).

[18] The Objective. La industria de los videojuegos ya genera más ingresos que la música y el cine juntos. *Periódico digital*, 2023. Visitado desde el 12 de agosto de 2023. (page 4).

[19] Unity. *Object Pooling* dentro de *Unity*. *Sitio Web*, 2024. Visitado desde el 13 de junio de 2024. (page 14).

[20] Unity Technologies. *Unity*. *Sitio Web*, 2023. Visitado desde el 6 de agosto de 2023. (page 26).

[21] M. P. Wolf. *Encyclopedia of Video Games: The Culture, Technology, and Art of Gaming*. *ABC-CLIO*, pages 6–7, 2012. ISBN 9780313379369. (page 4).