Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención en Tecnologías de la Información

Emulación del sistema de combate del dispositivo D-Tector para Android

Autor:
D. Diego Villegas Acero

Tutor:
D. Joaquín Adiego Rodríguez

Agradecimientos:

Me gustaría agradecer a mi familia por el apoyo incondicional que siempre me han proporcionado. En especial a mi madre, que siempre ha estado dispuesta a ayudar, y mi primo Adrián, que me ha acompañado durante toda mi vida, incluyendo esta etapa en la universidad, y en quien siempre he podido confiar. También me gustaría agradecer a los compañeros con los que he desarrollado una bonita amistad con la que compartir no solo aficiones sino también conocimiento. Por último me gustaría también agradecer a mis amigos de siempre Álvaro, Miguel y Rodrigo, con los que continúo teniendo la mejor de las amistades, y seguro leerán esto con ganas. Un enorme saludo y gracias por estar siempre ahí.

Resumen:

El objetivo del proyecto es realizar una adaptación a dispositivos móviles de un antiguo dispositivo electrónico portátil pensado para el entretenimiento, lanzado y distribuido por Bandai entre los años 2002 y 2003. Dicho dispositivo estaba diseñado para su fácil uso en exteriores y utilizaba algunas mecánicas de peleas bastante inusuales para la época, utilizando un sensor de movimiento para generar los distintos tipos de ataques. Ésta adaptación se centrará en recrear el sistema de combate original, intentando mantener la estética digital y retro del dispositivo que también le caracterizaba, haciendo uso del motor de videojuegos Unity.

Abstract:

The objective of the project is to create an adaptation to mobile devices of an old portable electronic widget designed for entertaining purposes, launched and distributed by Bandai between 2002 and 2003. The device was designed to be easily used outdoors and used quite unusual fighting mechanics for that time. This adaptation will focus on recreating the original battle system, trying to keep the digital and retro aesthetic of the device which also made it stand out, using Unity game engine.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1: Motivaciones y Objetivos

4.1 Motivaciones

El sector de los videojuegos lleva siendo tremendamente popular en las últimas décadas, y pese a la pandemia CoVid-19 sufrida hace unos años, parece que esta tendencia no va a la baja, sino al contrario. Según la web epdata [1] se ha visto potenciada, posiblemente por las restricciones externas sufridas dificultando el ocio en exteriores, y parte del sector dedicada a los dispositivos móviles ha generado un aumento en uso e ingresos en los últimos años [2].

El proyecto surge con el objetivo de familiarizarme con el popular motor de videojuegos Unity para acercarme más a este sector que siempre me ha apasionado, y además recrear el dispositivo que pude disfrutar durante mi niñez, actualizándole a los nuevos tiempos desarrollándolo para dispositivos móviles Android.

4.2 Objetivos

Comprender las distintas herramientas que ofrece Unity para la creación de videojuegos 2D para smartphones y aprender a hacer un uso correcto de éstas para el desarrollo de la aplicación móvil.

Implementación del sistema de combate utilizado en el videojuego original, manteniendo el sistema de ataque triangular estilo piedra-papel-tijera que lo caracterizaba, y el uso de la cámara que incluye la mayor parte de dispositivos móviles para generar los distintos tipos de ataque del juego a través de patrones.

Mantener la estética retro que utilizaba la consola mediante el uso de sprites con estética pixelart.

Dar un cierto valor estratégico al juego, ajustando estadísticas de los personajes y funcionalidades que permitan pensar estrategias para incrementar las posibilidades de victoria.

Capítulo 2: Contexto

2.1 Uso de la cámara en smartphones

Hay muchas aplicaciones para smartphone que utilizan la cámara para múltiples funciones, como la detección de movimiento para aplicaciones de vigilancia y seguridad [3], el escáner de códigos QR [4] o la monitorización de latidos del corazón mediante el uso de la cámara para detectar la palpitación sanguínea [5]. Sin embargo, las funcionalidades más comunes en el sector de los videojuegos son las relacionadas con la Realidad Aumentada, que permiten añadir dinamismo al juego, al permitir la interacción con el entorno en el propio juego. El caso más famoso es el de Pokemon GO [6], que se convirtió en el líder del mercado móvil durante varios años con su lanzamiento.

2.2 Los juegos de estrategia por turnos

Los videojuegos de estrategia por turnos [7] son un género de videojuegos en el que el avance sobre éste está divido por rondas o turnos. Estos videojuegos tienen una fase analítica que permite al usuario pensar en qué acción realizar sin que el tiempo utilizado afecte en el juego.

En este tipo de videojuegos es muy común el uso de técnicas y procedimientos que generen ventajas y desventajas sobre el usuario al asignar atributos o valores específicos sobre distintas variables como las armas, el terreno, o los personajes. Estas técnicas, al combinarse con otros valores y variables, aumentan el valor estratégico del juego sin necesidad de que sean difíciles de entender para el usuario.

Un ejemplo de estas técnicas podría ser la tabla de tipos elementales utilizada a lo largo de la saga de videojuegos Pokémon, en la cual los ataques y las unidades controlables por el jugador en combate (también llamados Pokémon), tienen asignado algún tipo elemental. Los elementos interactúan entre sí generándose estados de ventaja y desventaja táctica según el tipo del Pokemon aliado y enemigo, y los distintos tipos de ataque de los que disponga cada uno.

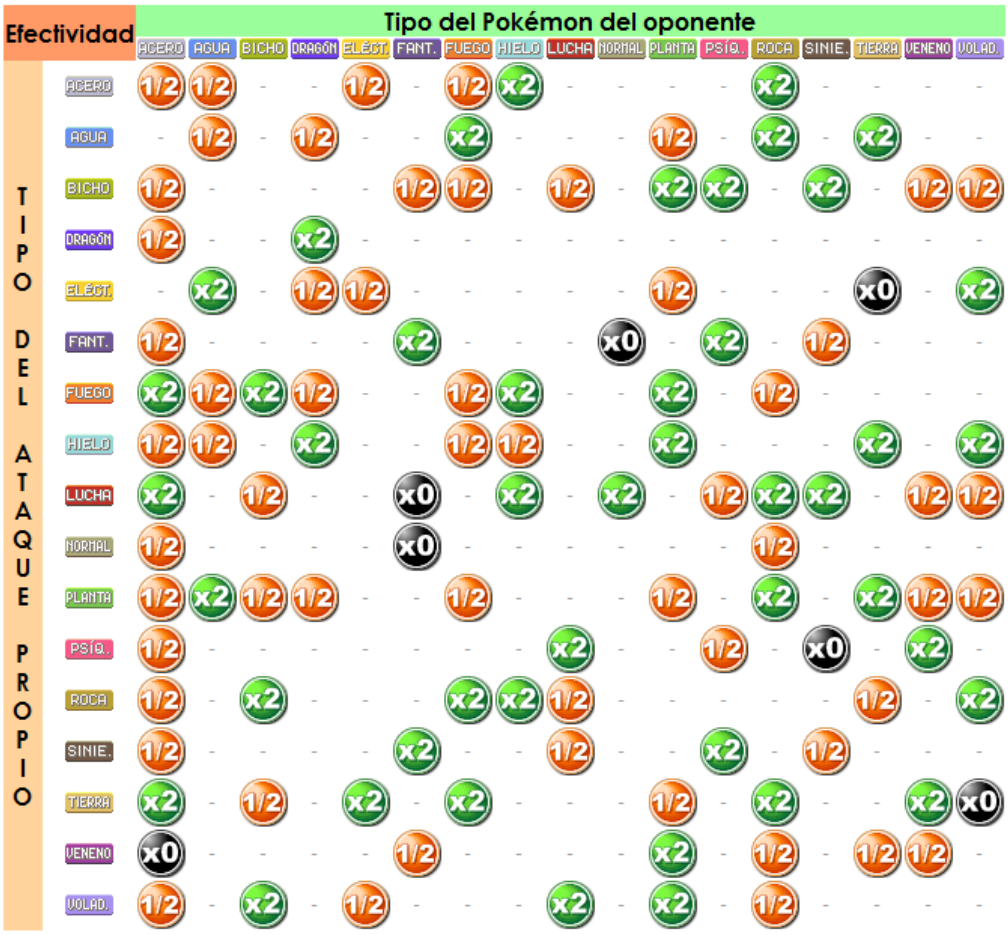

Ilustración 1: Tabla de tipos elementales presente en todos los videojuegos de la saga Pokémon

Sin embargo, técnicas tan desarrolladas como esta mencionada tabla, en combinación con las otras técnicas pueden resultar demasiado complicadas para el usuario, y es por eso que en la propia saga, el juego suele comenzar utilizando unidades de tipo planta, agua y fuego para que el usuario se familiarice con este valor estratégico poco a poco. La razón es que estos tipos generan un factor cíclico de ventaja-desventaja, de forma que el usuario aprende poco a poco el valor que tiene el conocimiento de los tipos elementales en el juego.

Este concepto estratégico que forma un grafo cíclico dirigido, reducido a la forma de un triángulo, actúa de forma similar al juego del piedra-papel-tijera y es utilizado de maneras más o menos complejas en múltiples juegos cuando se pretende añadir cierto valor estratégico.

Fire Emblem es otra de estas sagas en las que el valor estratégico tiene una gran importancia, y que introducen el concepto estratégico de ataque en triángulo. En algunos videojuegos de la saga se optaron por variaciones más complejas de este denominado triángulo de armas, como en el caso del videojuego Fire Emblem: Radiant Dawn [8] en el que existe un triángulo para las armas físicas, y un triángulo para las armas mágicas, el cual incluye otro triángulo interno.

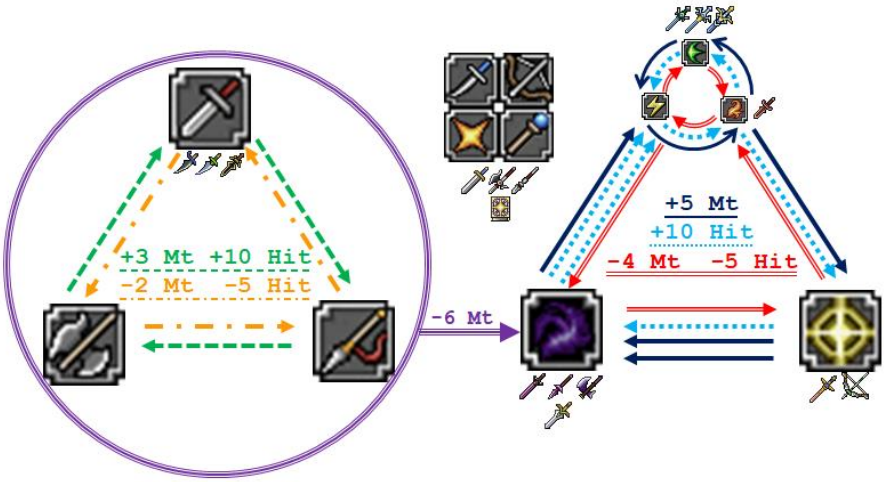

Ilustración 2: Triángulo de armas detallado del videojuego Fire Emblem: Radiant Dawn

Por otro lado, existen otros juegos de la saga, en los que se optó por variaciones más simples, como es el caso del videojuego Fire Emblem: Heroes [9], un videojuego para smartphones que se lanzó al mercado a comienzos del año 2017 y que utiliza la popular estrategia triangular de la manera más sencilla posible. Como en todos los videojuegos de la saga, existen múltiples armas y magias, pero a diferencia de los anteriores videojuegos, en éste las unidades están vinculadas a un tipo específico de arma, a la cual se la asigna un color. De esta forma, aunque una unidad pueda usar distintas armas, todas pertenecerán al mismo tipo, que se puede reconocer solamente por el color que tiene asignado. Haciendo muy fácil de entender si la unidad está en ventaja o desventaja frente a otra.

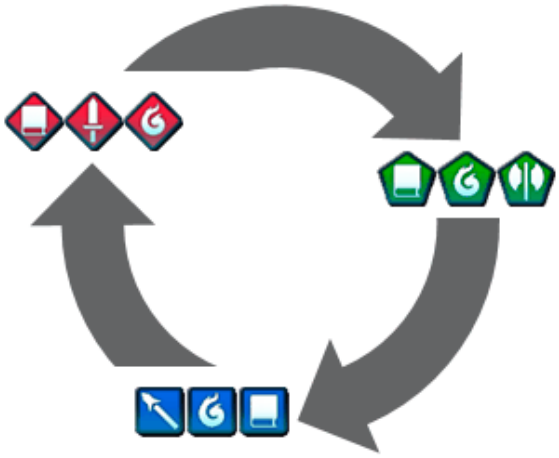

Ilustración 3: Triángulo de armas sencillo de Fire Emblem: Heroes

La franquicia Digimon tiene gran cantidad de videojuegos, y muchos de ellos podrían considerarse del género de estrategia por turnos. En la mayoría de estos juegos, como en los mencionados anteriormente, se aplica el concepto estratégico mencionado anteriormente, usando atributos y tipos que forman grafos cíclicos dirigidos, y que generalmente tienen esta forma triangular para evitar complejidad.

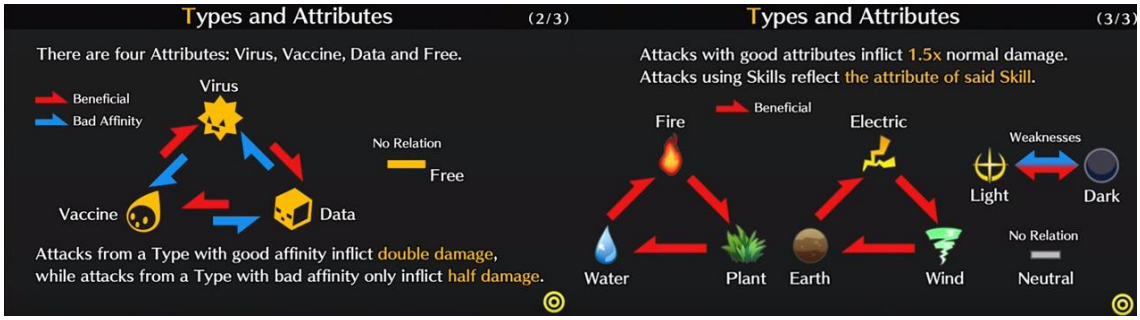

Ilustración 4: Imagen mostrando la relación entre elementos y atributos en el videojuego Digimon Story Cyber Sleuth

2.3 Los dispositivos Digivice

Los Digivice [10] son una línea de juguetes pertenecientes a la saga Digimon (Digital Monsters), basados en los aparatos surgidos en las distintas series de TV de la saga. Su significado es un diminutivo de Digital Device, y su funcionalidad estaba únicamente vinculada al ocio.

La idea de estos dispositivos surge como una evolución de otra línea de productos de la marca, los V-Pet, los cuales eran muy similares a los populares Tamagotchis, cuya finalidad del juego era criar, cuidar y entrenar a un ser digital, considerando a los portadores de estos dispositivos entrenadores de estas mascotas.

Ilustración 5: Imagen de las distintas versiones V-Pet originales

Con la popularización del anime Digimon Adventure, surgieron estos nuevos dispositivos llamados a partir de entonces Digivice, los cuales, a diferencia de las anteriormente mencionadas V-Pet, se centraban más en el combate, ajustándose más a la idea del anime y manteniendo la idea de la aventura, ya que incluían contadores de pasos para avanzar en el juego.

El éxito de la serie original fomentó la emisión de nuevas temporadas de la franquicia en televisión, las cuales permitieron que continuaran surgiendo nuevos dispositivos con distintas funcionalidades para cada modelo nuevo y siempre con funcionalidades en sintonía con su correspondiente serie televisiva. Así surgió el Digivice denominado D-Tector (D-Scan en Japón), en el que se basa este proyecto.

Ilustración 6: Imagen de la caja en la que iba incluido el dispositivo D-Tector

Las funcionalidades que caracterizaban a este dispositivo en particular era el uso de un sensor de movimiento para múltiples funcionalidades dentro del juego, especialmente en el combate. En el combate se realizaba un ataque por turno, existiendo 3 tipos de ataque que funcionaban de una manera similar al piedra-papel-tijera, y seleccionando cuál se quería utilizar al realizar pasadas de la mano sobre el sensor usando ciertos patrones sencillos.

Esto añadió un cierto valor estratégico al juego, ya que las estadísticas de daño no se compartían con cada ataque, por lo que se debía elegir el tipo en consideración al monstruo que estuvieras usando en el momento, y al rival al que te enfrentaras.

Capítulo 3: Planificación

3.1 Introducción

En este capítulo se hablará sobre el método de planificación utilizado para el proyecto y se evaluarán varios riesgos considerados que pudieran afectar al desarrollo de éste, además del método utilizado para aliviar o resolver por completo los problemas que estos riesgos puedan llegar a provocar en el proyecto.

3.2 Método de planificación

El método utilizado para el desarrollo del proyecto se basa en el denominado Proceso Unificado de Desarrollo Software [11], el cual se centra en dividir el proyecto en casos de uso fundamentales, y aplicar un desarrollo en espiral sobre ellos.

Esto es, por cada tarea identificada como fundamental en el proyecto, se realiza un proceso iterativo e incremental, de manera que se pasan por las distintas fases del desarrollo para todas estas tareas múltiples veces, añadiendo nuevos requisitos y depurando posibles fallos en los distintos ciclos del proceso. Estos ciclos pueden realizarse de manera paralela para los distintos casos de uso, de manera que no es necesaria ni conveniente la completa depuración de una tarea para para poder avanzar con el resto.

De manera simplificada, las fases de trabajo por las que pasan las distintas tareas son la obtención de requisitos, el análisis del programa o la tarea en cuestión, el diseño de éste, en el que se organiza la información recopilada, la implementación y por último la prueba o debugging.

Conociendo estas distintas fases de trabajo, se pueden enumerar las siguientes etapas por las que pasa un proyecto desarrollado mediante el Proceso Unificado:

- **Fase de inicio:** Durante esta fase se define el alcance que tendrá el proyecto, es decir, las distintas tareas o casos de uso que se deben desarrollar y completar para dar el proyecto por concluido. Para ello es necesaria la recopilación de información sobre los distintos recursos y conocimientos que puedan ser necesarios para llevar a cabo el proyecto.
- **Fase de elaboración:** Aquí se llevan a cabo posibles rectificaciones sobre los casos de uso propuestos en la fase de inicio y se define una estructura para el proyecto. El objetivo de esta fase es tener un plan del proyecto de manera clara, para poder seguirlo a lo largo de las otras fases sin cambios de requisitos.

- **Fase de construcción:** Esta fase suele ser la más larga del proyecto ya que incluye toda la implementación de los casos de uso definidos anteriormente. Al final de esta fase deben quedar completos los distintos casos de uso del proyecto, consiguiendo una versión funcional del proyecto.
- **Fase de transición:** Tras llegar a una versión funcional del proyecto se procede a la distribución de este para que el usuario pueda probar el programa en el entorno especificado en los casos de uso anteriormente definidos. En esta fase también se pueden elaborar planes sobre posibles nuevas funcionalidades del programa para el futuro.

Todo esto permite que el proyecto pueda avanzar de manera uniforme en las distintas áreas de las que se compone sin generar bloqueos en el avance del proyecto.

3.3 Planificación del proyecto

El estar trabajando al mismo tiempo que desarrollando este proyecto y teniendo aún una asignatura pendiente, hace que resulte complicado una planificación por fechas precisa, dado que el tiempo empleado será inferior en los primeros meses del proyecto hasta que acabe con la asignatura en cuestión, pero sí es posible una planificación en horas para las distintas fases del proyecto.

La idea y visto bueno de este proyecto ha sido dado a la última semana de Noviembre de 2021, desde la cual se comenzó con el avance del proyecto.

Aunque no resulte muy precisa, supondré una media de 10 horas semanales para dedicar en el proyecto de las cuales la mayor parte se realizarán de Viernes a Domingo, sin embargo los primeros meses la media será probablemente menor. Por otro lado, pasado el mes de Febrero y superada la asignatura, la media de horas será posiblemente algo mayor, por lo que la media puede llegar a ser orientativa aunque no sea altamente precisa.

Fase	**Duración estimada**	**Semana aproximada de inicio**	**Semana aproximada de finalización**
Inicio	40 horas	22/28 de Noviembre 2021	13/19 de Diciembre 2021
Elaboración	50 horas	13/19 de Diciembre 2021	17/23 de Enero 2022
Construcción	200 horas	17/23 de Enero 2022	6/12 de Junio 2022
Transición	10 horas	6/12 de Junio 2022	13/19 de Junio 2022

Tabla 1: Tiempo estimado para la realización del proyecto por fases

La razón de que la fase de construcción sea tan duradera se debe a que Unity es una herramienta nueva para mí, por lo que la aparición de dificultades a la hora de implementar el programa es bastante probable.

Tampoco se abandonan las fases de inicio y elaboración, ya que precisamente por ser una herramienta sobre la que no hay experiencia personal, es necesaria una documentación inicial considerable para conocer cómo obtener los requisitos del proyecto y el análisis de éste.

Por otro lado la fase de transición estimo que tenga una duración más reducida, dado que la herramienta es conocida por resultar fácil de utilizar para múltiples plataformas, por lo que la inclusión a los dispositivos considerados debería realizarse sin mucha dificultad.

3.4 Gestión de riesgos

La aparición de riesgos a lo largo de un proyecto es algo muy común, pero es importante analizar la posibilidad de aparición de estos y el impacto que pueden generar en el tiempo, ya que será necesario plantear soluciones ante estos problemas para reducir el impacto en el proyecto.

Mediante el uso de una tabla, podemos categorizar los riesgos según su importancia basándonos en la probabilidad de aparición y el impacto en el proyecto que implicaría la aparición de éste. De esta forma podemos prestar una atención discriminada ante los riesgos según su importancia.

Probabilidad\Impacto	**Crítico**	**Grave**	**Moderado**	**Despreciable**
Alta	Máxima	Máxima	Media	Baja
Media	Máxima	Alta	Baja	Baja
Baja	Alta	Alta	Baja	Nula
Muy baja	Media	Media	Baja	Nula

Tabla 2: Tabla de Probabilidad/Impacto de los riesgos

A continuación concreto las distintas categorías comentadas en la tabla anterior mediante el uso de métricas para ofrecer una información más completa del análisis:

Categorías según la probabilidad:

- Alta: Probabilidad de aparición mayor 60%.
- Media: Probabilidad de aparición de entre el 60% y 35%.
- Baja: Probabilidad de aparición de entre el 34% y el 15%.
- Muy baja: Probabilidad de aparición inferior al 15%.

Categorías según el impacto en tiempo y/o recursos que generarían su aparición:

- Crítico: Impacto en al menos un 30% del coste original.
- Grave: Impacto de entre un 29 y un 20% del coste original.
- Moderado: Impacto de entre un 19 y un 10% del coste original.
- Despreciable: Impacto en menos de un 10% del coste original.

Teniendo clara la esta clasificación, listamos los riesgos identificados:

Riesgo	**R1. Estimación de tiempos del proyecto inferiores a los reales**
Descripción	Considerar tiempos para fases del proyecto inferiores al tiempo real necesario para la finalización de estas
Importancia	Máxima
Consecuencia	Retraso en la fecha de finalización del proyecto
Estrategia	Evitar el riesgo
Plan de contingencia	Aumentar el tiempo dedicado al proyecto para mantener una fecha de finalización similar a la original

Tabla 3: Riesgo 1 del proyecto

Riesgo	**R2. Errores en la implementación de funcionalidades del programa**
Descripción	Los recursos utilizados para la implementación de alguna funcionalidad no funcionan como se esperaba
Importancia	Máxima
Consecuencia	El programa no será funcional
Estrategia	Aceptar el riesgo
Plan de contingencia	Realizar una adaptación o rediseño del programa para conseguir que vuelva a ser funcional, alterando los requisitos de la menor manera posible

Tabla 4: Riesgo 2 del proyecto

Riesgo	**R3. Problemas en el software utilizado**
Descripción	Errores o incompatibilidades debido a los distintos recursos software utilizados para el desarrollo del proyecto
Importancia	Baja
Consecuencia	Retraso en el avance del proyecto
Estrategia	Aceptar el riesgo
Plan de contingencia	Buscar el origen del error y buscar recursos software alternativos si fuera necesario

Tabla 5: Riesgo 3 del proyecto

Riesgo	**R4. Problemas de hardware**
Descripción	Problemas hardware de los distintos dispositivos utilizados de manera habitual para el desarrollo del proyecto
Importancia	Nula
Consecuencia	Reemplazo de los dispositivos fallidos
Estrategia	Aceptar el riesgo
Plan de contingencia	Usar un equipo portátil para reemplazar al usado de manera habitual en caso de fallo de este

Tabla 6: Riesgo 4 del proyecto

Riesgo	**R5. Problemas de entorno**
Descripción	Problemas en el lugar habitual de trabajo que impidan o dificulten el avance del proyecto en dicho lugar
Importancia	Nula
Consecuencia	Búsqueda de un nuevo entorno de trabajo
Estrategia	Aceptar el riesgo
Plan de contingencia	Tener un equipo portátil disponible para poder trabajar desde distintos entornos fácilmente

Tabla 7: Riesgo 5 del proyecto

Riesgo	**R6. Pérdidas de progreso del proyecto**
Descripción	Pérdidas del trabajo realizado en el proyecto por fallos externos al desarrollador
Importancia	Baja
Consecuencia	Reconstruir las pérdidas del proyecto
Estrategia	Reducir el riesgo
Plan de contingencia	Mantener copias del proyecto en un segundo equipo para evitar grandes pérdidas

Tabla 8: Riesgo 6 del proyecto

Riesgo	**R7. Indisponibilidad del desarrollador**
Descripción	Problemas personales de algún tipo que afecten al desarrollador e impidan el avance en el proyecto
Importancia	Baja
Consecuencia	Retraso en el avance del proyecto
Estrategia	Aceptar el riesgo
Plan de contingencia	Ajustar los tiempos dedicados al proyecto cuando se vuelva a estar disponible

Tabla 9: Riesgo 7 del proyecto

3.5 Costes del proyecto

Se ha realizado una estimación del presupuesto que requerirá el desarrollo del proyecto considerando que éste es realizado por un autónomo, utilizando los conocimientos obtenidos en las asignaturas de la carrera Planificación y Gestión de Proyectos [12] y Fundamentos de Organización de Empresas [13]. Con esto en mente, se ha desglosado el presupuesto del proyecto en distintos tipos de costes:

3.5.1 Costes hardware

En los costes hardware se incluyen los distintos dispositivos necesarios para la realización del proyecto, añadiendo una estimación del uso que se les da a partir de la planificación descrita anteriormente.

Hardware	**Coste aproximado de compra**	**Tiempo de uso estimado**	**Tiempo de uso total estimado**	**Coste de uso**	**Coste de mantenimiento**
Ordenador sobremesa	800€	5 horas por día según la planificación	60 días	0,085 x 60 = 5,1€	0,22 x 60 = 13,2€
Portátil de repuesto	600€	0.5 horas cada 3 días para generar copias de seguridad	2 días	0,085 x 2 = 0,17€	0,17 x 2 = 0,34€
Móvil Redmi9	150€	0.5 horas por día para pruebas	6 días	0,005 x 3 = 0,015€	0,08 x 6 = 0,48€

Tabla 10: Costes hardware del proyecto

El hardware anteriormente mencionado no se compró específicamente para este proyecto, sino que ya se disponía anteriormente de él, y puede utilizarse para futuros proyectos, por lo que no deberá tenerse en cuenta el coste de compra para el cálculo del presupuesto del proyecto sino únicamente el coste de mantenimiento y uso del equipamiento siendo el total del proyecto de unos 19,31€. Téngase en cuenta que el coste se calcula únicamente por el uso en el proyecto, a pesar de que estos dispositivos también son utilizados de manera externa a este.

Para calcular el coste de uso de los PCs se utiliza una media del coste eléctrico [14] por uso del aparato[15], mientras que para el móvil se utiliza la media del coste de una carga completa del dispositivo [16] y una estimación de las cargas usadas para el tiempo utilizado en el proyecto.

El coste de mantenimiento se calcula a partir de una estimación del tiempo de vida útil del aparato y el uso de este en el proyecto.

3.5.2 Costes software

El software utilizado para la realización del proyecto es totalmente gratuito, sin embargo, para algo más profesional se consideraría el uso de software Premium para mejorar el resultado final del producto.

Software	**Coste de compra aproximado**
Software de detección de movimiento OpenCV for Unity	84,87€
Unity Pro	1.800€/año
Recursos gráficos y sonoros de la Unity Asset Store	200€

Tabla 11: Costes software del proyecto

Si se deseara comercializar del producto final, se trataría de crear sprites nuevos y abandonar la recreación del producto original, dado que sería necesaria una licencia comercial para el uso de los sprites de la marca que se pretende recrear. Y dado que dicha marca sigue en funcionamiento hoy en día la obtención de una licencia comercial para el uso de esta sería totalmente inviable. Esto se tiene en cuenta en el apartado software de Recursos de la Unity Asset Store, aunque no necesariamente dichos recursos se obtuvieran en esta plataforma en concreto, ya que podrían realizarse personalmente para el proyecto por personal externo pagado a comisión por los recursos.

3.5.3 Costes de personal

El coste por recursos gráficos y sonoros se ha considerado como software de la tienda de recursos de Unity, por lo que este apartado incluye únicamente el coste por el tiempo empleado por el desarrollador, y los gastos del entorno de trabajo.

Por otro lado, para este proyecto no se ha realizado ninguno de los siguientes gastos, sino que se tratan de una estimación de lo que podrían ser unos gastos realistas

Razón	**Coste medio estimado**	**Coste total estimado en el proyecto**
Salario del desarrollador de la aplicación	35.000€/año = 19,19€/h	19,19 x 300 = 5.757€
Alquiler de oficina personal	350€/mes	350 x 8 = 2.800€

Tabla 12: Costes de personal del proyecto

Para el cálculo de salario se ha investigado sobre el salario bruto medio anual de un desarrollador de aplicaciones móviles [17]. Teniendo en cuenta esto, se ha realizado una estimación de los días laborales del 2022 en trabajos de jornada completa para obtener un salario aproximado en horas, y así obtener el coste del proyecto.

El alquiler del lugar se ha calculado a partir de una aproximación a la media de varios anuncios de alquiler de oficina en Valladolid, considerando aquellos que incluyen el mínimo material de oficina necesario para el proyecto, y que la oficina no necesita tener unas dimensiones superiores a 80m^2. El coste del alquiler, no se corresponde con el gasto que implicaría el proyecto en una jornada laboral de 8 horas, ya que se ha seguido la planificación inicial de 5 horas diarias durante 3 días a la semana. Por lo que el alquiler se amplía a 8 meses.

Capítulo 4: Análisis

4.1 Introducción

En el capítulo de análisis se mostrará el trabajo de diseño de requisitos y casos de uso de la aplicación previo al desarrollo e implementación de ésta. Se dividirán los requisitos según el tipo al que pertenecen y se explicará que es lo que les caracteriza para que pertenezcan a dicho tipo. También se desarrollarán los casos de uso, explicando los posibles pasos a seguir en los distintos escenarios.

Dado que este trabajo se realiza anteriormente al desarrollo final de la aplicación, pueden surgir ligeros cambios en los requisitos y esquemas desarrollados a continuación.

4.2 Actores y roles

El proyecto tendrá únicamente un actor, que tendrá el rol de cliente, ya que no existen funcionalidades exclusivas según el rol. El objetivo es que la aplicación sea homogénea para el usuario que la utilice, independientemente de quién sea. Esto permite que el desarrollador pueda ver el punto de vista del cliente final en la aplicación fácilmente, ya que no hay otros usuarios que interfieran en el uso de la aplicación, únicamente las acciones del propio usuario afectan a la respuesta que genera el programa.

4.3 Definición y clasificación de requisitos

Definir los requisitos de la aplicación ayuda a identificar y segmentar las tareas de la aplicación. Para tener una clasificación de requisitos más ordenada y según aprendimos en las asignaturas centradas en Diseño e Ingeniería Software, se segmentarán los requisitos en funcionales y no funcionales siguiendo los apuntes de la asignatura Modelado de Software [18]. A continuación se describirán los distintos requisitos clasificándolos según sus características.

4.3.1 Requisitos funcionales

Los requisitos funcionales son aquellos que generan una interacción entre el usuario y el sistema software, generando servicios y restricciones en el comportamiento del programa. A continuación se nombrarán los requisitos de usuario considerados para el proyecto, incluyendo también los requisitos funcionales de información considerados.

Requisito	Descripción
RF1	El usuario podrá seleccionar la dificultad del combate
RF2	El usuario podrá elegir la acción a realizar en cada turno del combate
RF3	El usuario podrá elegir el tipo de ataque a realizar en cada turno
RF4	El usuario podrá abandonar el combate en cualquier momento
RFI1	El sistema deberá guardar las distintas estadísticas del aliado y los distintos enemigos: vida, fuerza, velocidad y técnica
RFI2	El sistema deberá almacenar el ataque seleccionado por el usuario en cada turno: energy, rush o ability

Tabla 13: Lista de requisitos funcionales del proyecto

4.3.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que definen propiedades emergentes del sistema. Estos requisitos incluyen elementos estéticos, de seguridad y de las decisiones del desarrollador al implementar los requisitos funcionales.

Requisito	Descripción
RNF1	El programa se realizará utilizando el motor Unity
RNF2	El programa utilizará el lenguaje C# para la implementación de scripts
RNF3	El programa deberá poder ejecutarse en un dispositivo Android
RNF4	El sistema deberá cargar los combates en menos de 0,5 segundos
RNF5	El sistema deberá mostrar al usuario las animaciones correspondientes a los ataques ejecutados en el combate
RNF6	El sistema deberá mostrar al usuario la vida de los personajes cada vez que ésta cambie a lo largo del combate
RNF7	El sistema utilizará la cámara para asignar el ataque realizado por el usuario
RNF8	El sistema indicará cuándo detecta movimiento con la cámara durante la selección del ataque
RNF9	El sistema mostrará un contador antes de comenzar la detección de movimiento de la cámara
RNF10	El sistema mantendrá una estética retro lo más similar posible a la del dispositivo original
RNF11	El sistema deberá cargar a un enemigo distinto según la dificultad
RNF12	El sistema tendrá un patrón de ataque diferente para cada enemigo
RNF13	El sistema deberá permitir realizar la acción de curación solo una vez por combate
RNF14	El sistema deberá limitar la curación para que sobrepase la vida máxima del jugador
RNF15	El sistema no deberá mostrar números de vida negativos
RNF16	El sistema deberá restablecer los valores de vida al reiniciar cada combate
RNF17	El sistema generará un ataque aleatorio cuando el seleccionado por el usuario no se ajuste con ninguno de los establecidos

Tabla 14: Lista de requisitos no funcionales del proyecto

4.4 Casos de uso

Los casos de uso describen las distintas acciones que los usuarios pueden realizar en el sistema junto con la secuencia de acciones que dicha elección generan en el sistema.

A partir de los requisitos del sistema desarrollados anteriormente se puede desarrollar un esquema de casos de uso. Los distintos casos de uso se centrarán en los requisitos funcionales, sin embargo, es necesario tener en cuenta también los distintos requisitos no funcionales para la especificación de estos.

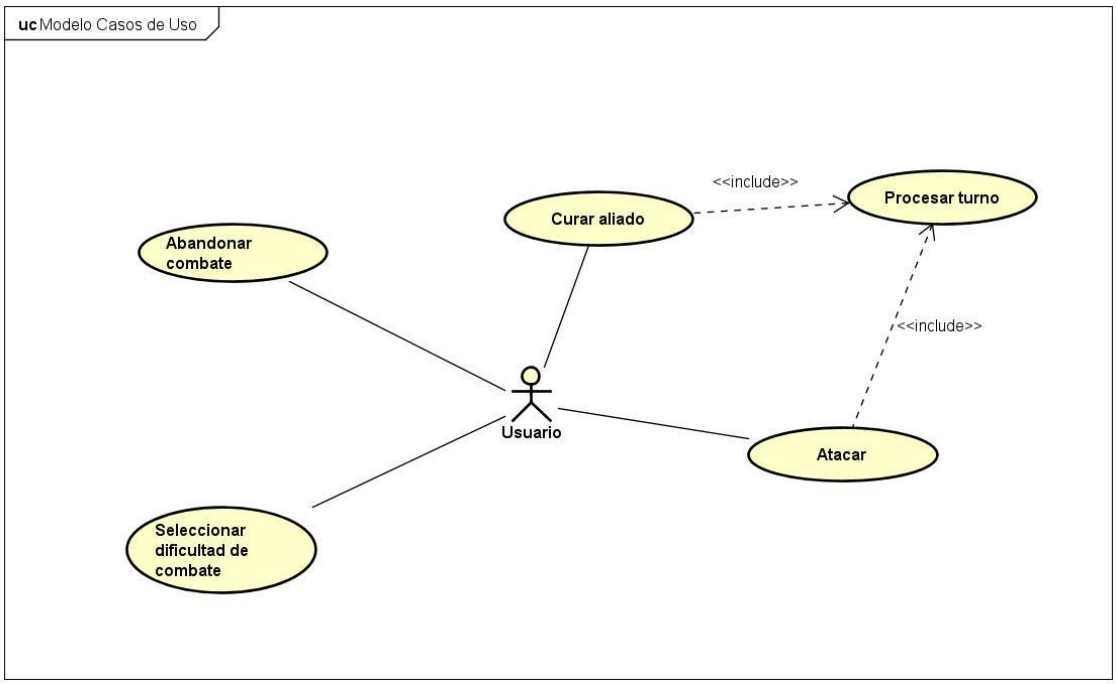

Ilustración 7: Diagrama de Casos de Uso

Tras tener un diagrama de los casos de uso, se pueden detallar éstos para obtener una idea más detallada del funcionamiento seguido por el sistema. A continuación se realizará esta especificación de casos de uso, en la que se detallarán las acciones que conllevan los distintos casos de uso paso a paso, y detallando las secuencias alternativas en cada caso, si existen.

En estas tablas deberían incluirse el actor/es que protagonizan el caso de uso específico, pero dado que en la aplicación del proyecto solo existe un actor “usuario” se omitirá esta información redundante.

CU1	**Seleccionar dificultad de combate**
Precondición	No hay
Secuencia normal	1. El usuario pulsa el botón correspondiente a la dificultad que desea
	2. El sistema almacena la opción elegida
	3. El sistema carga el escenario de combate
	4. El sistema carga al enemigo correspondiente según la opción elegida anteriormente por el usuario
	5. El sistema realiza las animaciones de inicio de combate
	6. El sistema carga el menú de combate
Secuencia alternativa	No existen secuencias alternativas posibles
Postcondición	No hay

Tabla 15: Secuencia del CU "Seleccionar dificultad de combate"

CU2	**Atacar**
Precondición	El usuario ha seleccionado una dificultad de combate y el sistema ha cargado el menú de combate
Secuencia estándar	1. El usuario pulsa el botón de atacar
	2. El sistema muestra una cuenta atrás
	3. El sistema activa la cámara cuando termina la cuenta atrás
	4. El usuario tapa el sensor de la cámara
	5. El sistema guarda la información obtenida
	6. El sistema comprueba la información y guarda el ataque a realizar
	7. El sistema carga el escenario de combate y se realiza el caso de uso “CU5 - Procesar turno”
Secuencia alternativa	3'. Si no se detecta la cámara del dispositivo el sistema guarda un ataque aleatorio y el caso de uso continúa en el paso 7
	4'. Si el usuario no tapa el sensor de la cámara en un tiempo de 1 segundo, el caso de uso continúa en el paso 5
	6'. Si el sistema no tiene suficiente información para asignar el ataque a realizar, el caso de uso continúa en el paso 3
Postcondición	Se guarda el tipo de ataque que realizará el aliado en el turno

Tabla 16: Secuencia del CU "Atacar"

CU3	**Curar aliado**
Precondición	El usuario ha seleccionado una dificultad de combate y el sistema ha cargado el menú de combate
Secuencia estándar	1. El usuario se desplaza por el menú para encontrar la opción de curar
	2. El usuario pulsa el botón de curar
	3. El sistema carga el escenario de combate y se realiza el caso de uso “CU5 - Procesar turno”
Secuencia alternativa	No existen secuencias alternativas posibles
Postcondición	No hay

Tabla 17: Secuencia del CU "Curar aliado"

CU4	**Abandonar combate**
Precondición	El usuario ha seleccionado una dificultad de combate y el sistema ha cargado el menú de combate
Secuencia estándar	1. El usuario se desplaza por el menú para encontrar la opción de escapar
	2. El usuario pulsa el botón de escapar
	3. El sistema carga el menú de inicio
Secuencia alternativa	No existen secuencias alternativas posibles
Postcondición	Se eliminan los datos del combate y la dificultad guardados

Tabla 18: Secuencia del CU "Abandonar combate"

CU5	**Procesar turno**
Precondición	Se ha seleccionado una opción de combate (el tipo de ataque a realizar, o la curación)
Secuencia estándar	1. El sistema comprueba la acción a realizar por el aliado
	2. El sistema muestra el ataque realizado por el aliado mediante la animación correspondiente
	3. El sistema muestra el ataque realizado por el enemigo con su correspondiente animación
	4. El sistema muestra al personaje que perdió el turno
	5. El sistema muestra la vida restante del personaje mientras se actualiza con el resultado del turno
	6. El sistema carga el menú de combate
Secuencia alternativa	1'. Si la acción a realizar es la curación del aliado, el sistema muestra al personaje aliado y el caso de uso continúa en el paso 5
	5'. Si la vida del personaje en cuestión es 0, el sistema muestra una animación correspondiente a la victoria/derrota del jugador, y carga el menú de inicio. A continuación el caso de uso queda sin efecto
Postcondición	La vida de los personajes no supera el máximo del personaje ni es negativa

Tabla 19: Secuencia del CU "Procesar turno"

4.5 Modelo de Dominio inicial

En esta sección se desarrollará un modelo de dominio inicial para la aplicación. Debe aclararse que es posible que surjan cambios en este modelo, dado que el conocimiento propio sobre la plataforma de desarrollo Unity es actualmente escaso, y pueden surgir problemas con el esquema propuesto, por lo que debe usarse únicamente para tener una idea sobre la propuesta del proyecto, y no como una guía o arquitectura exacta de la aplicación final.

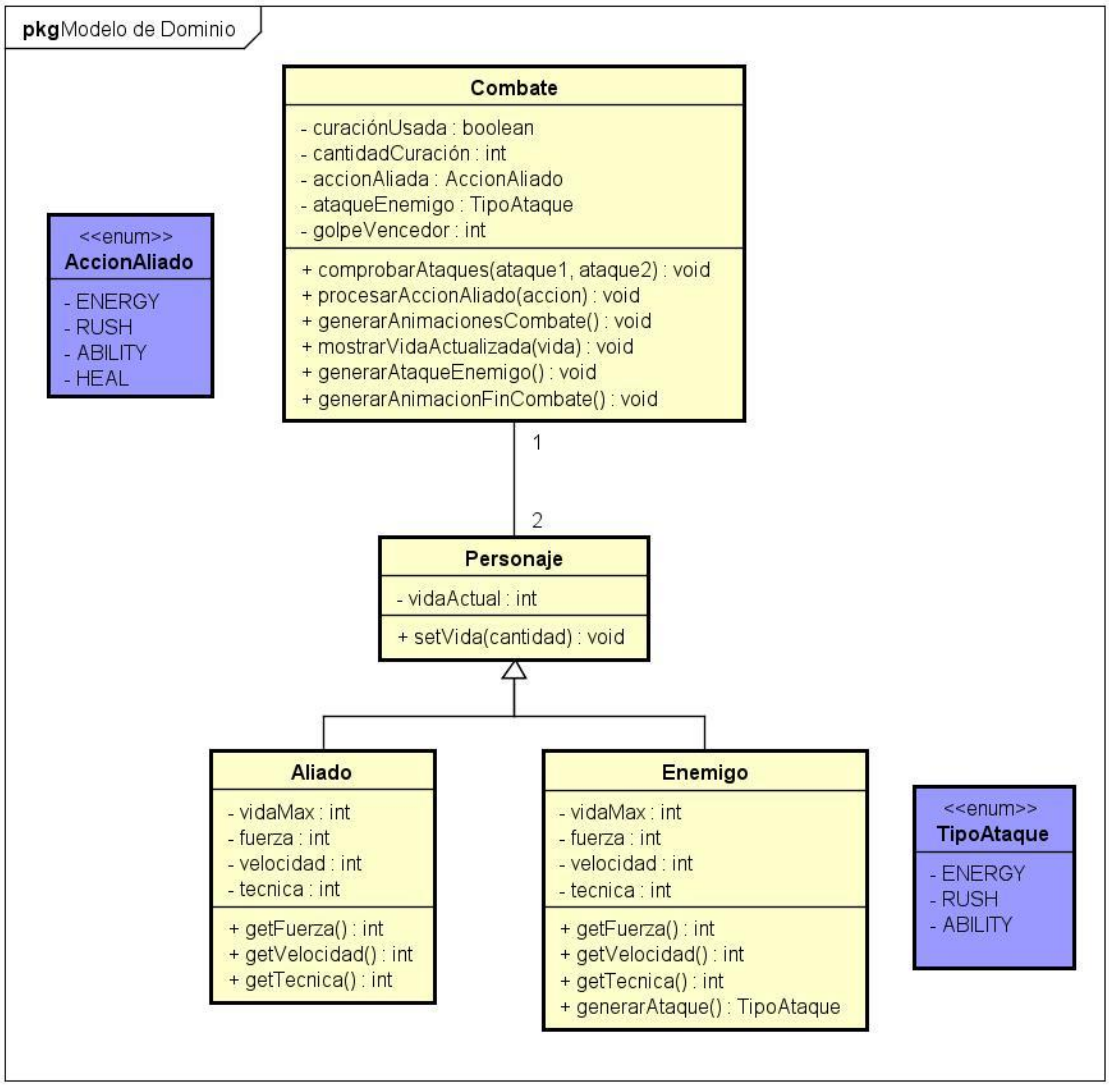

Ilustración 8: Modelo de dominio inicial de la aplicación

En este esquema podemos ver las distintas clases que existirán en la aplicación junto con la relación de multiplicidad entre ellas. También se pueden ver las variables que considero que serán necesarias para el proyecto y a las funciones necesarias para el desarrollo de la aplicación.

4.6 Diagramas de secuencia

Teniendo en cuenta el modelo de dominio propuesto desarrollaré la secuencia de operaciones e interacción que se generará entre el usuario y la aplicación en los distintos casos posibles.

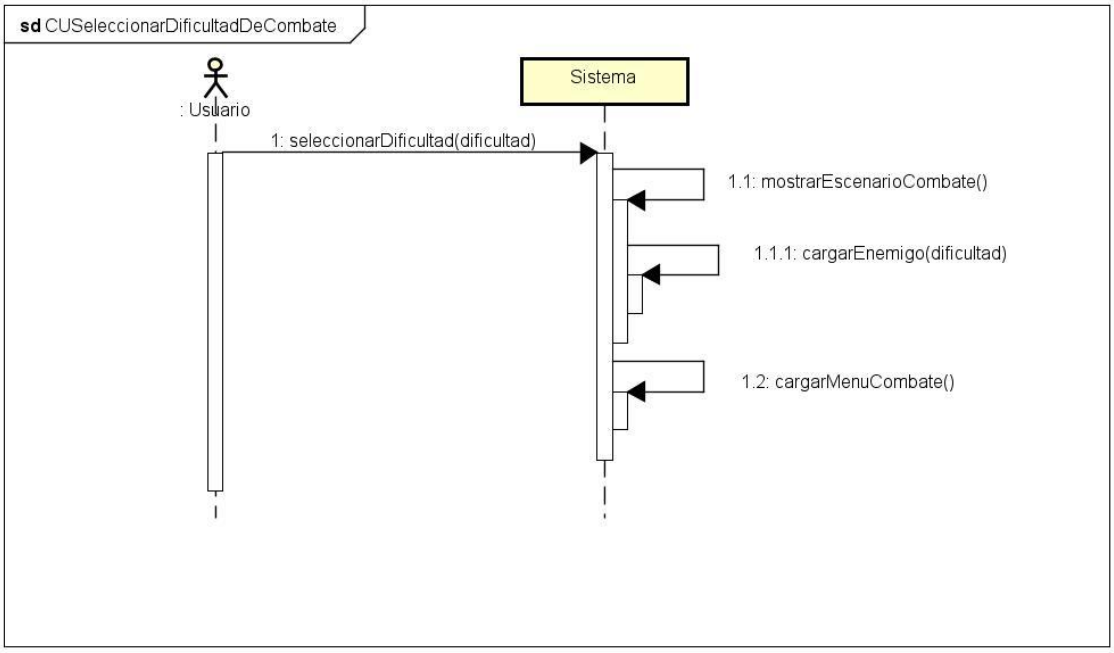

Ilustración 9: Diagrama de secuencia del CU "Seleccionar dificultad de combate"

Este diagrama muestra la interacción de operaciones para generar el combate correspondiente a la dificultad seleccionada por el usuario. Para mostrar las opciones de dificultad se utilizará un menú sencillo, ya que esta parte no formaba parte del dispositivo original que se pretende recrear.

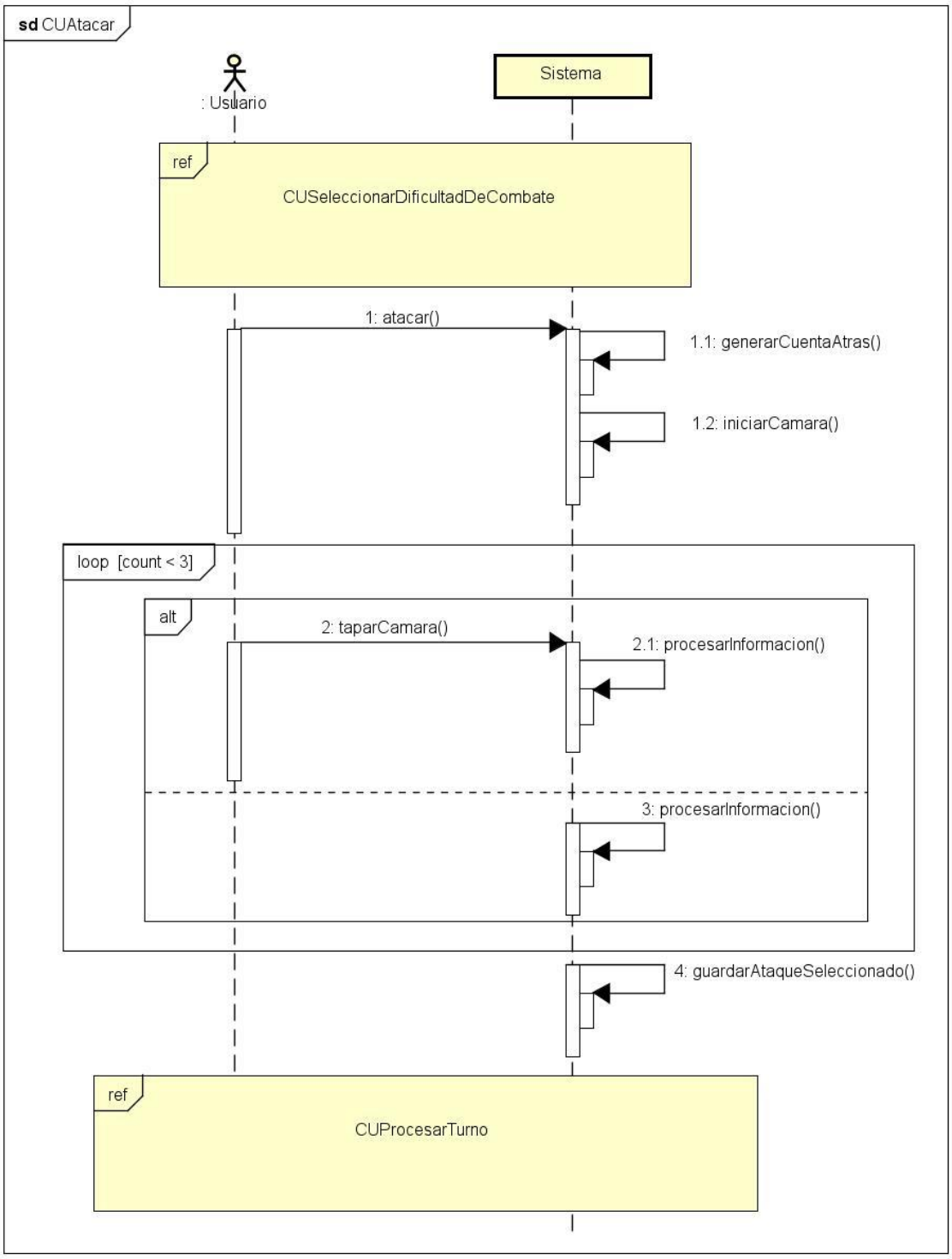

Ilustración 10: Diagrama de secuencia del CU "Atacar"

Como se puede ver, en este diagrama se incluye el CU Seleccionar dificultad de combate a pesar no verse implementada una relación entre estos CU en el diagrama de CU realizado anteriormente.

La razón es que para realizar el CU Atacar, es necesario que se haya cargado un combate, pero no puede mostrarse con una relación de inclusión (include) del CU Seleccionar dificultad de combate, ya que no es la única opción posible. Tampoco es correcto añadirlo con una relación de extensión (extend), ya que son casos de uso independientes, y una vez realizado el CU Seleccionar dificultad de combate se puede realizar éste y otros CU sin que intervenga de nuevo la selección de dificultad.

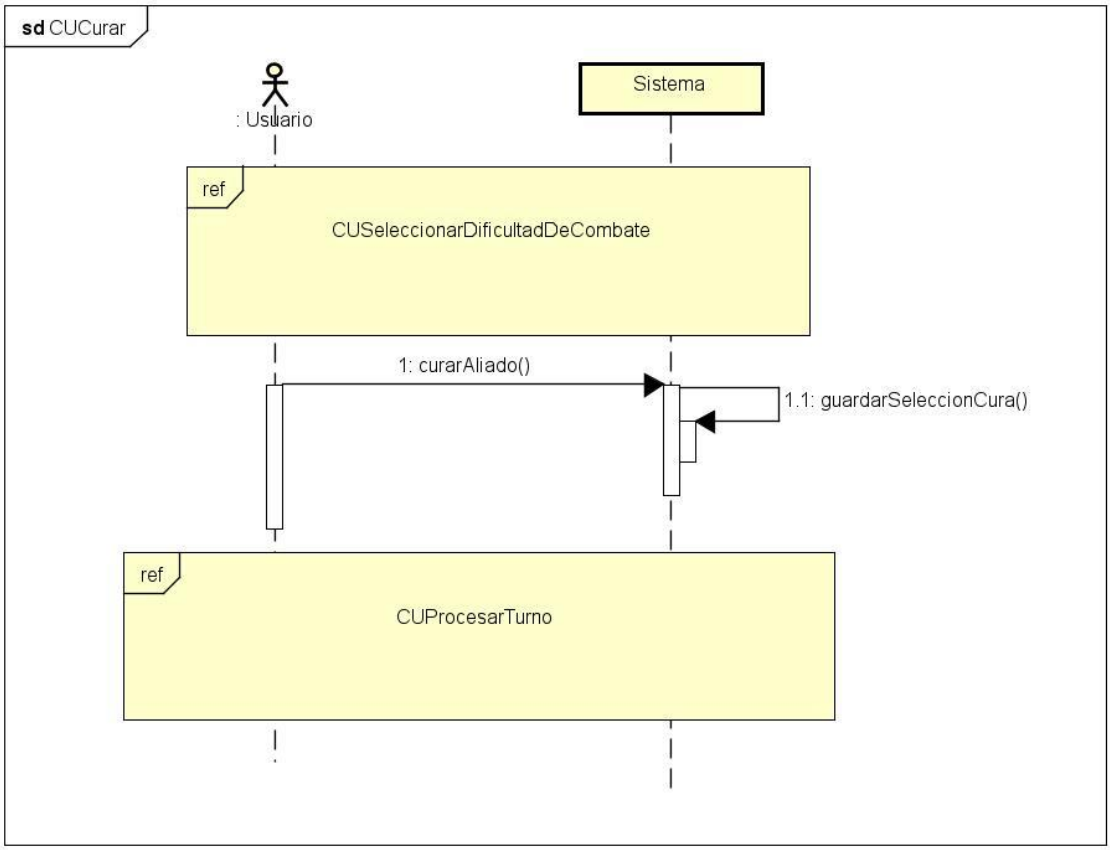

Ilustración 11: Diagrama de secuencia del CU "Curar aliado"

Este CU es similar al CU Atacar, en ambos se muestra la inclusión Procesar turno, que es una extensión de éstos.

A continuación se muestra la secuencia de operaciones correspondiente al CU Procesar turno. Pese a utilizar la terminología CU, no se trata de un CU real, ya que el usuario no interactúa con el sistema, sino de una inclusión de otros CUs. Ha de tenerse en cuenta que, ya que es la extensión de estos casos de uso, no hay una interacción con el usuario. Simplemente se ha añadido como un caso de uso independiente para facilitar la visualización de éstos.

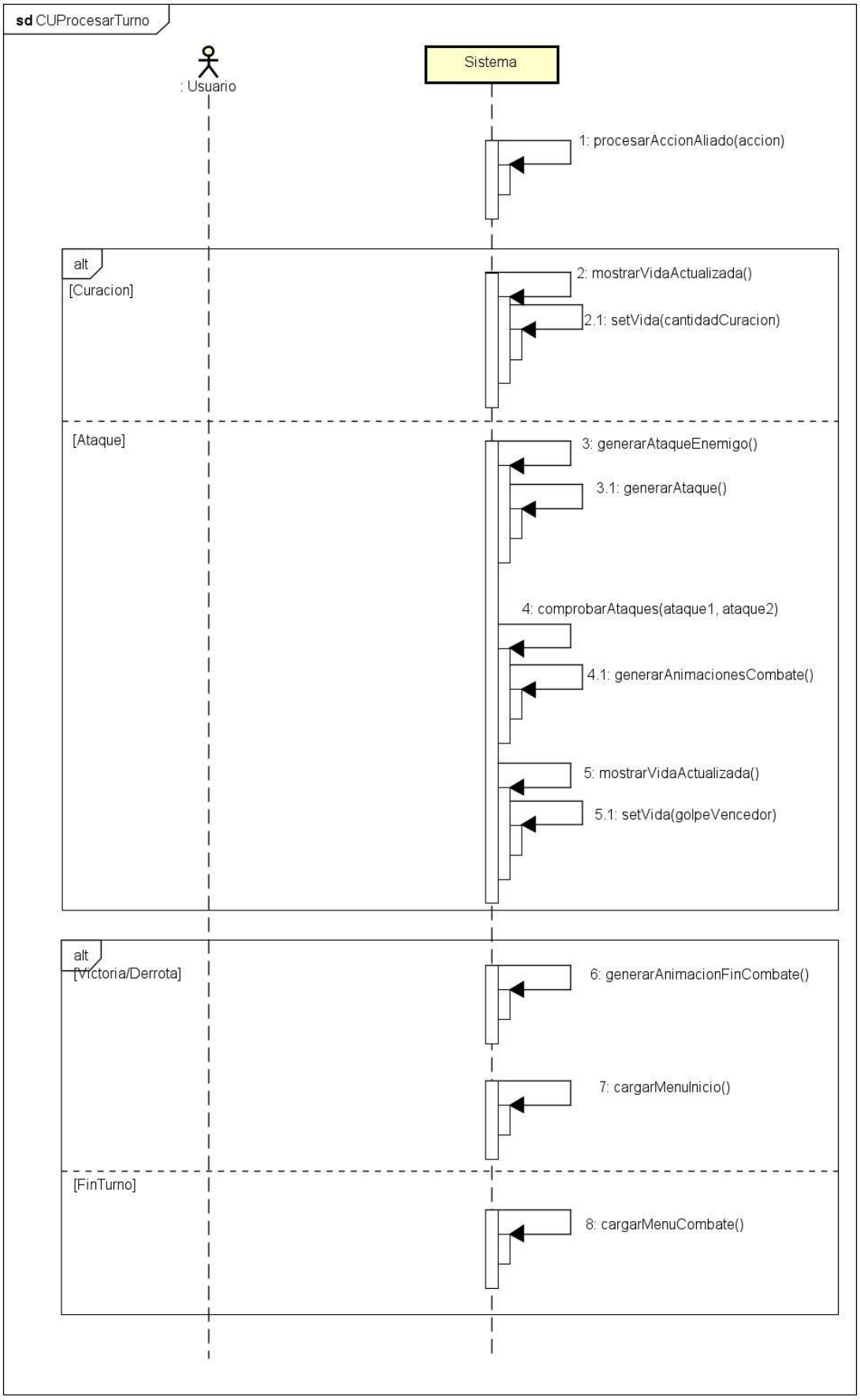

Ilustración 12: Diagrama de secuencia del CU "Procesar turno"

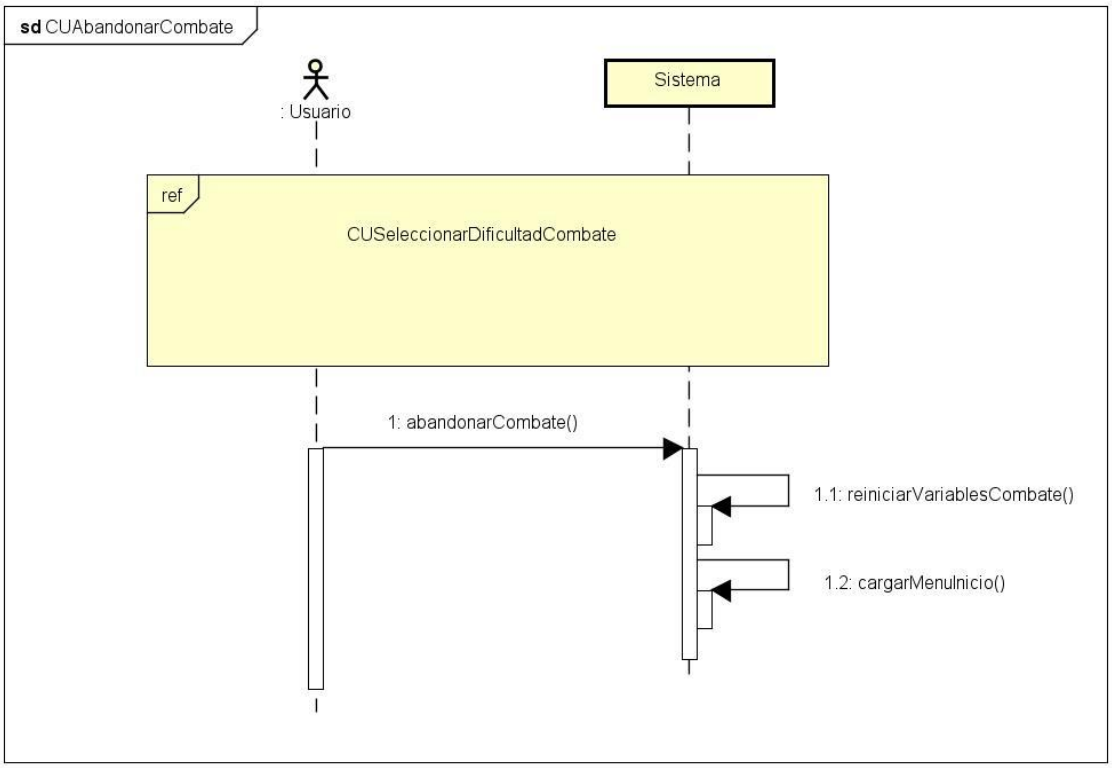

Ilustración 13: Diagrama de secuencia del CU "Abandonar combate"

A través de estos dos últimos casos de uso mostrados, es posible volver al CU Seleccionar dificultad de combate, ya que pueden cargar el menú inicial del juego, donde se hace la elección de la dificultad.

Capítulo 5: Diseño y arquitectura

5.1 Introducción

En este capítulo describiré las distintas decisiones de desarrollo tomadas en la creación del proyecto, describiendo la arquitectura del motor utilizado, y cómo ésta ha afectado a distintas decisiones en el diseño del proyecto. También se incluirá el diseño arquitectónico final del proyecto, el cual ha cambiado ligeramente respecto del propuesto anteriormente para adaptarse a la plataforma de desarrollo.

5.2 Arquitectura de la aplicación

Esta sección mostrará la estructura de la aplicación, comenzando por la del propio motor Unity, y finalmente por la del proyecto realizado.

En la estructura del motor de videojuegos, centraré principal atención en aquellas partes que mayor uso he dado, ignorando aquellos aspectos que no he utilizado.

5.2.1 Arquitectura Unity

Unity utiliza una arquitectura Entity Component System (ECS) [19], una arquitectura bastante común en desarrollo de videojuegos basada en un principio utilizado en el Diseño de Software [20], que consiste en fomentar la composición por encima de la herencia.

Esto es, la combinación de objetos (componentes) para crear nuevas entidades y así obtener nuevas funcionalidades, de forma que una entidad no se ve restringida en su funcionalidad por el tipo de objeto que es, como ocurriría utilizando herencia, puesto que la funcionalidad de la entidad variará si se añaden, eliminan o sustituyen componentes que lo formen, aunque el tipo del objeto se mantenga igual.

Conociendo cómo se conforman y comportan las entidades y componentes bajo esta arquitectura, falta por conocer el significado de sistema. En una arquitectura ECS se denomina sistema al proceso que maneja la interacción entre las entidades y los componentes. A pesar de que pueda resultar ambiguo por la definición de la palabra, los motores que utilizan esta arquitectura no suelen disponer de un único sistema, sino de varios sistemas que manejan entidades con componentes que generen funcionalidades concretas. Un ejemplo podría ser un sistema de físicas, el cual maneja las entidades que disponen de las funciones necesarias para generar los cálculos físicos necesarios (masa del objeto, velocidad, posición…).

A continuación mostraré un diagrama de clases obtenido del TFG de Joaquín Casas Albertos [21] en el que se describe la estructura básica de las clases de Unity.

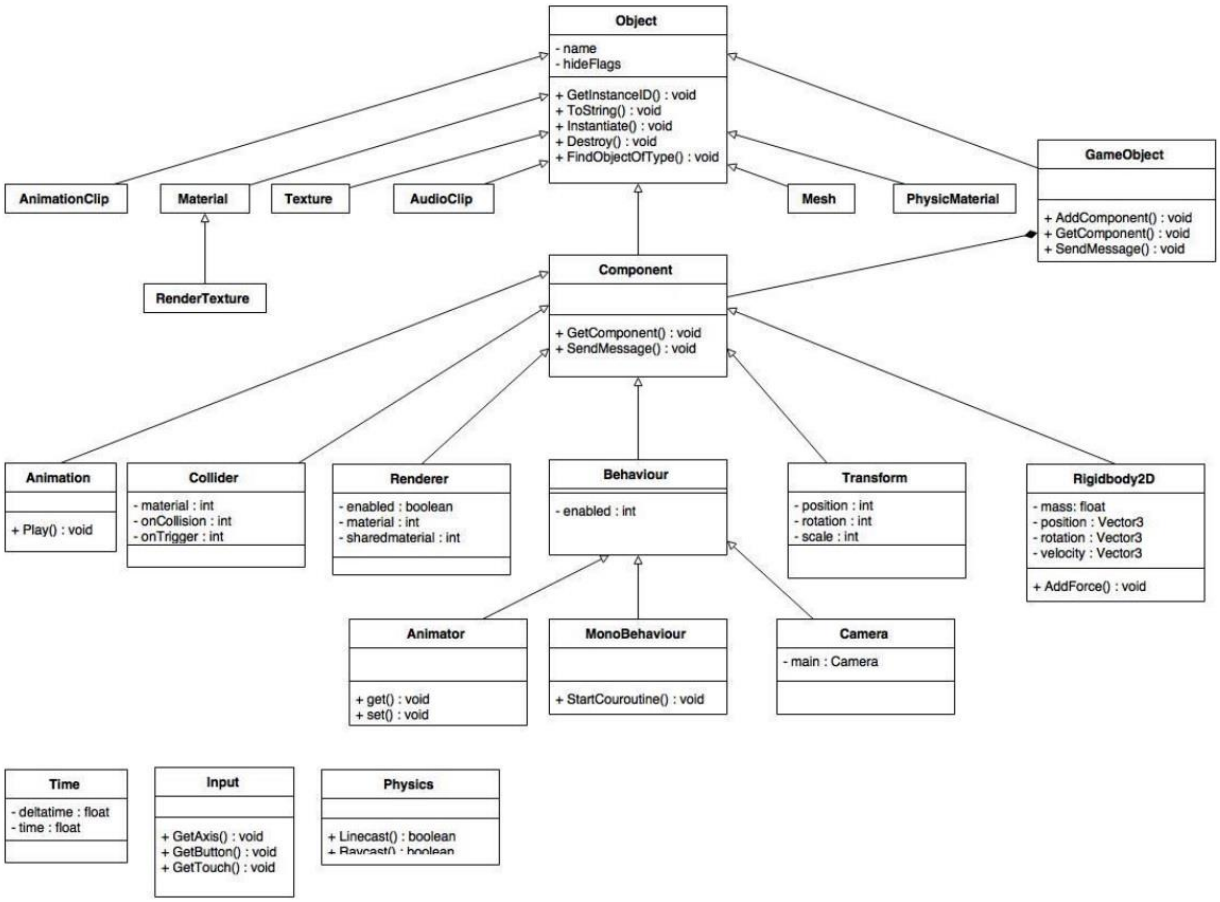

Ilustración 14: Modelo de dominio representando la estructura de Unity

En el modelo de dominio mostrado, se ven las principales clases que conforman la arquitectura de Unity. Las funciones y los atributos mostrados en cada clase no están completas, sino que muestran aquellos más comunes.

A continuación describiré las principales clases usadas en el proyecto:

- GameObject: Clases contenedor para crear instancias de objetos en tiempo de ejecución sobre las que se añaden clases Component. Para acceder a la mayoría de las funcionalidades del objeto se utiliza la función GetComponent<T>() sustituyendo T por el nombre de la clase Component que se quiere obtener.
- Component: Clase base para las distintas funcionalidades de un GameObject, y conformada por múltiples clases prediseñadas con nombre propio a los que se les denomina componentes. Estos componentes tienen las funcionalidades suficientes para implementar cualquier aplicación, ya que se pueden incluir Scripts personalizados y éstos son tratados como componentes.

- Renderer: La clase base para múltiples clases de renderizado y que permiten que los objetos, y en general elementos visuales, se muestren en la pantalla. Por ejemplo, durante este proyecto se ha hecho uso de la clase SpriteRenderer, que permite que se muestren las imágenes en la aplicación.
- Animator: Esta clase es la encargada de controlar las distintas animaciones creadas sobre los objetos, a través de funciones como setTrigger permite el cambio entre estados de animación.
- Camera: Esta clase se añade a un GameObject sobre el cual se mostrará la visión del usuario sobre la aplicación. Esto permite mantener objetos renderizados y al mismo tiempo fuera del campo de visión del usuario.
- Transform: Contiene las funciones y los atributos necesarios para obtener y modificar el posicionamiento del objeto sobre el que se añade, incluyendo la ubicación y la rotación del objeto. También permite generar animaciones al asignar un tiempo sobre el que se deber realizar el desplazamiento de los objetos.
- MonoBehaviour: Sobre esta clase se construyen todos los scripts personalizados para la aplicación

5.2.2 Arquitectura del proyecto

Conociendo la arquitectura del motor utilizado para el desarrollo y teniendo una idea del funcionamiento de las clases mostradas en el diagrama, detallaré la estructura de paquetes de alto nivel utilizada en la aplicación.

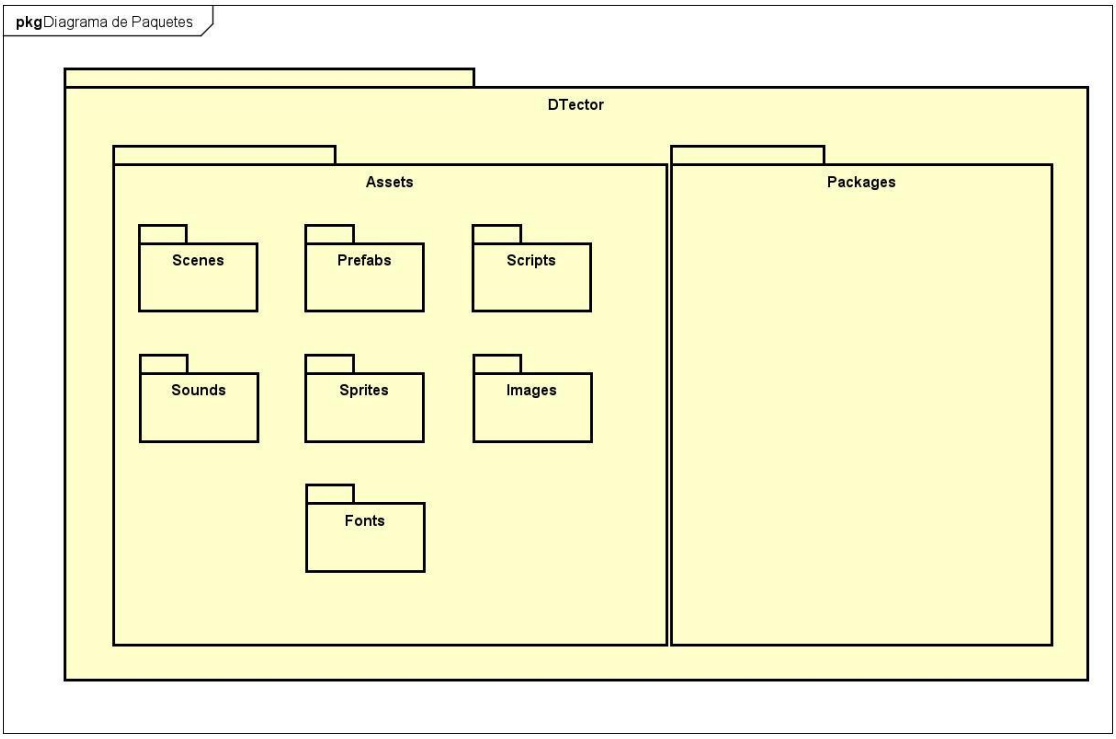

Ilustración 15: Diagrama de paquetes de alto nivel de la aplicación

El paquete Assets contiene todos los recursos y materiales utilizados para el desarrollo de la aplicación. La clasificación interna de ese directorio se realiza a gusto del usuario. Para mantener cierta organización se clasificaron los distintos recursos en:

- Scenes: Guarda las distintas escenas que debe cargar la aplicación.
- Prefabs: Almacena los distintos objetos que necesitan cargarse múltiples veces en una escena.
- Scripts: Incluye los distintos archivos con código personalizado que se utiliza en el programa. Tenga en cuenta que todo script en Unity es considerado un componente, y por tanto debe tener asociado un GameObject a él.
- Sounds: En este directorio se han organizado los distintos archivos de sonido utilizados para la aplicación.
- Sprites: Incluye los distintos materiales utilizados para la creación de los personajes y sprites más complejos. Esto incluye las imágenes/sprites, las animaciones y el animator asociado a esas animaciones. Dentro de este directorio se ha hecho otra clasificación para localizar más fácilmente a los personajes, ya que cada uno está organizado en un directorio propio dentro de Characters, de forma que si se quieren añadir nuevos personajes, se pueda hacer siguiendo la estructura del resto de personajes.
- Images: Almacena las imágenes de fondo y sprites sencillos que no necesitan animaciones, como las asociadas al menú de combate.
- Fonts: Aquí se guarda los archivos utilizados para la configuración de la fuente de texto utilizada en el programa.

El paquete Packages incluye los distintos archivos predeterminados del motor, incluyendo los distintos archivos y materiales utilizados para la interfaz gráfica predeterminada, además de librerías y archivos descargables desde la store de Unity (en caso de que se haya descargado alguno). La estructura interna de este directorio es algo compleja, por lo que se ha optado por no mostrarla en el diagrama, para mantener la claridad del esquema propio realizado.

A continuación mostraré un diagrama de paquetes más detallado en el que muestre la interacción entre estos.

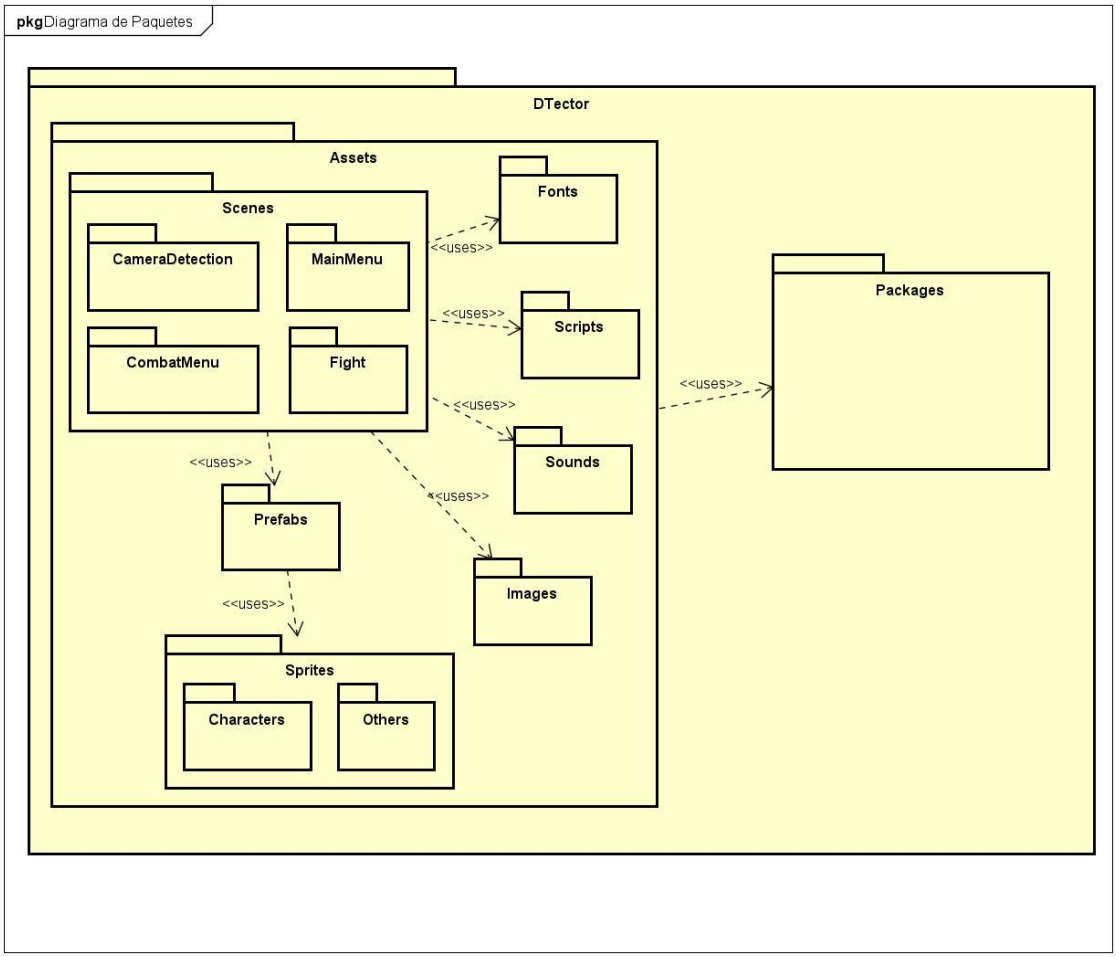

Ilustración 16: Diagrama de paquetes detallado

Conociendo la estructura del proyecto a nivel de paquetes, mostraré ahora la estructura interna a nivel de clases. Téngase en cuenta que dado que Unity utiliza la arquitectura ECS anteriormente comentada, lo que mostraré en realidad es la estructura de los scripts utilizados en la aplicación, y dichos scripts tendrán ligados un GameObject en el programa, los nombres utilizados han sido los de los objetos a los que estos scripts están ligados, por lo que muchas de estas clases se considerarán controladores que manejen y generen cambios en objetos o escenas.

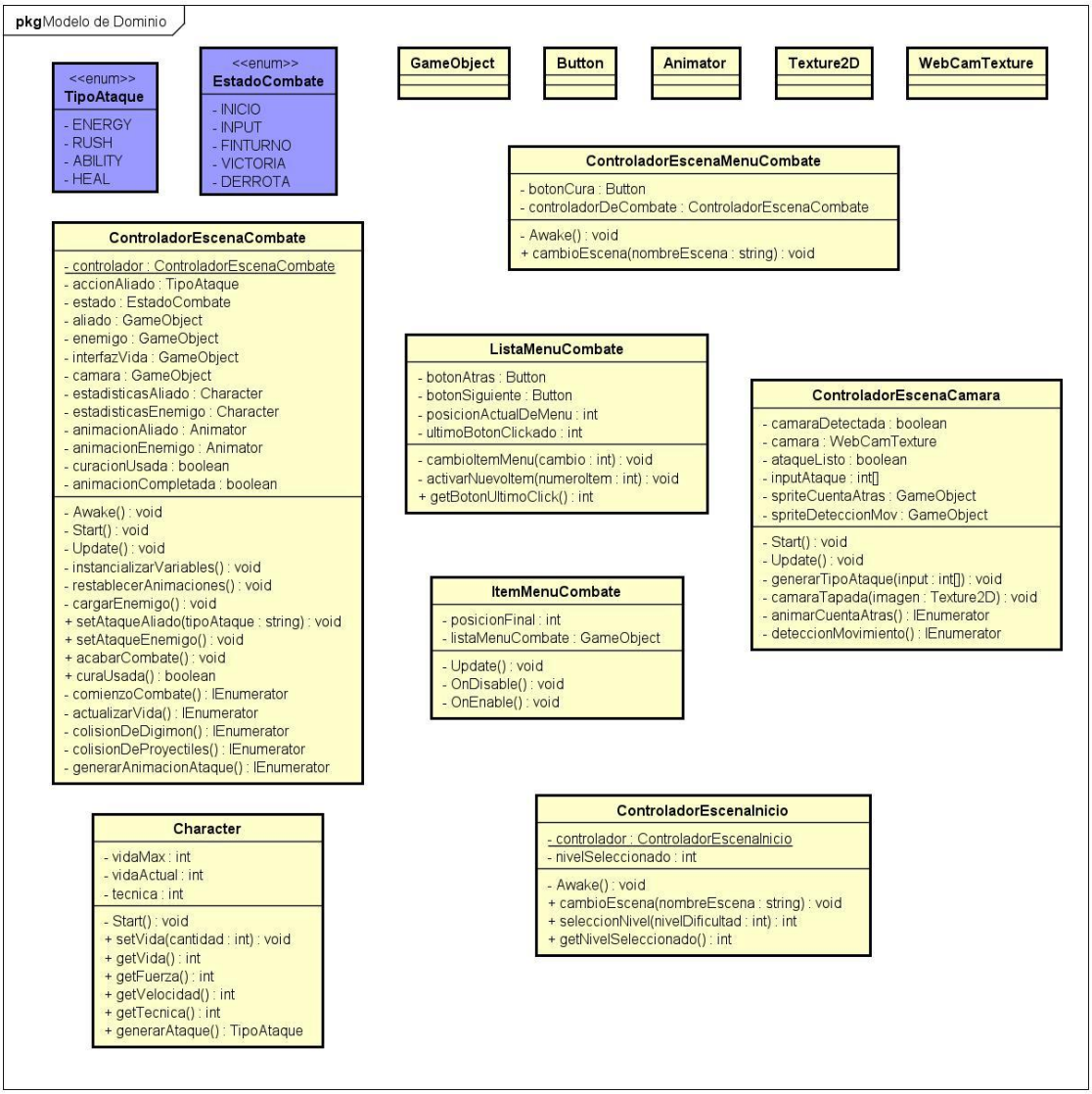

Ilustración 17: Diagrama de clases final de la aplicación

Si se observa detalladamente, se puede ver una clase IEnumerator como tipo de retorno de algunas funciones que no aparece en el diagrama. La razón es que no es un tipo como tal, sino la forma que he utilizado para representar las corrutinas del programa. Este tema se hablará con mayor detalle en el capítulo de Implementación.

Nota: Tras la realización del diagrama se hicieron pequeñas modificaciones para añadir algunas funcionalidades extra, por lo que en el diagrama faltan algunas de estas funciones. El cambio más notorio es la inclusión de probabilidades de cada tipo de ataque a los personajes, faltando dichos atributos y las funciones relacionadas con ellos en el diagrama.

5.3 Patrones de diseño utilizados

En esta sección se explicarán los principales patrones de diseño usados para el diseño de la aplicación y estudiados en la asignatura Diseño de Software [20], utilizando algunas figuras obtenidas de dicha asignatura para representar dichos patrones.

5.3.1 Patrón Singleton

Este patrón restringe la creación de instancias para clases concretas, es decir, permite asegurar que existe un único objeto de una clase. El objetivo de este patrón es facilitar el acceso a una instancia de manera global, al evitar la duplicidad de ésta.

Para implementar este patrón, se crea un método que genere una instancia de la propia clase si todavía no existe una y se restringe la creación de nuevas instancias reduciendo el acceso al método.

Este método fue necesario para evitar la duplicación de ciertos controladores en la aplicación, ya que era necesario mantenerlos activos entre los cambios de escena para evitar la pérdida de información. En el caso de esta aplicación, la creación se realiza en el método Awake de Unity, de forma que al generarse la escena, se comprueba que no existía un controlador para ésta, y de ser así, se destruye la instancia que se pretendía crear, dejando la original intacta y evitando que se reestablezcan los datos.

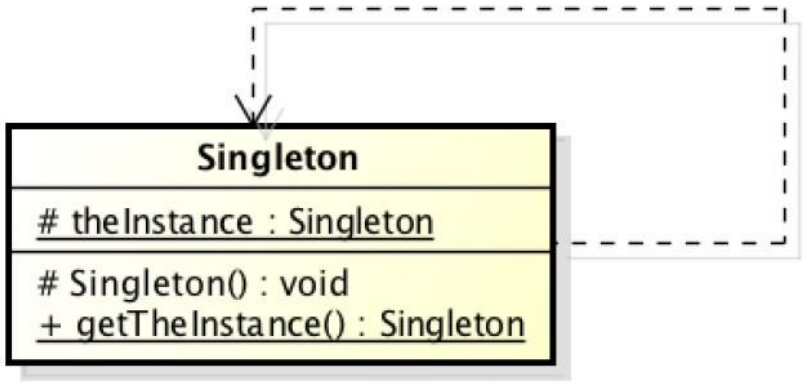

Ilustración 18: Representación del patrón Singleton

Como se puede ver, la clase contiene un constructor que comprueba si existe ya una instancia de la clase, y en caso de ser así evita su creación. En el caso del proyecto no fue necesaria la implementación de un método público get para obtener la instancia ya que se pueden obtener los GameObject asociados a un script mediante el método Find seguido del nombre del GameObject correspondiente.

5.3.2 Patrón Controlador

Este patrón forma parte de los patrones GRASP (General Responsibility Assignment Software Patterns) y consiste en delegar las tareas externas a un controlador, evitando que la interfaz realice las tareas asignadas a un evento, permitiendo localizar fácilmente la clase asociada a eventos y procesos.

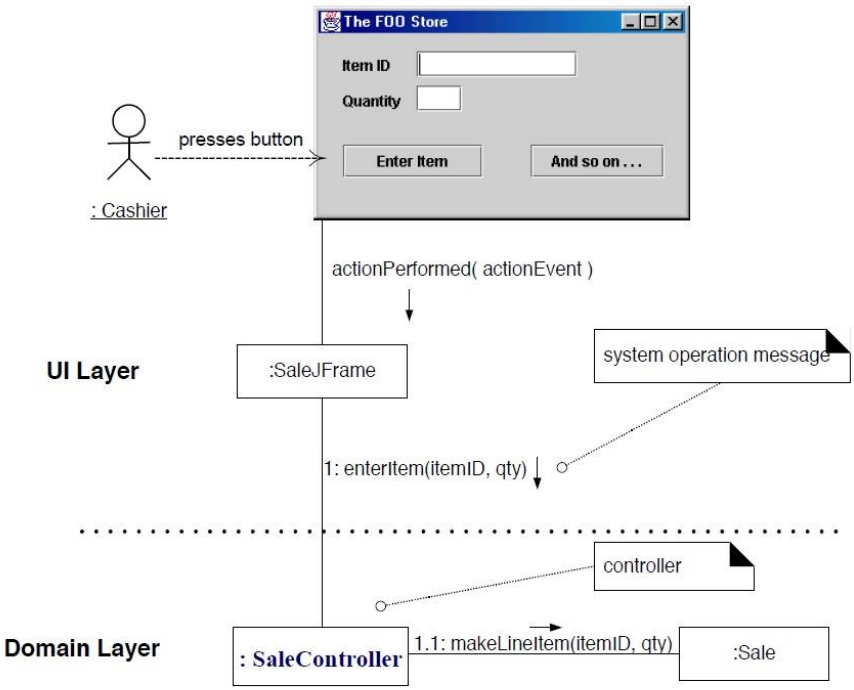

Ilustración 19: Ejemplo del patrón Controlador

Este patrón ha sido utilizado en gran parte del proyecto como se ha podido apreciar en el diagrama de clases, pero además, para evitar la saturación de un único controlador, se ha optado por la creación de controladores de escena, de forma que las funcionalidades queden correctamente agrupadas.

5.3.3 Patrón Experto

Otro patrón que forma parte de los patrones GRASP, es el patrón Experto. Éste consiste en asignar la funcionalidad de un evento a aquella clase que tenga la información necesaria para realizarla.

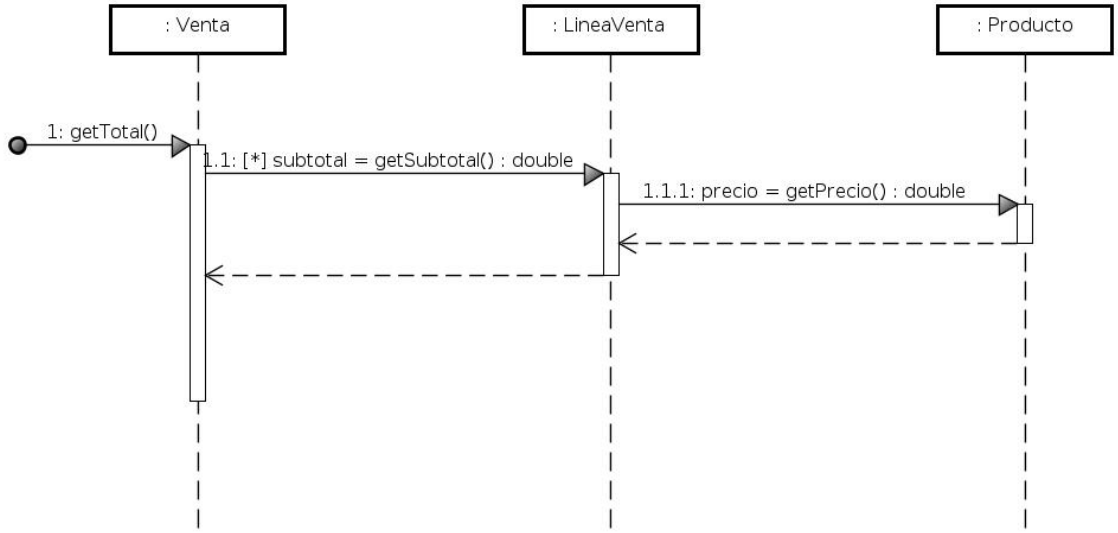

Ilustración 20: Ejemplo del patrón Experto

En el ejemplo se puede ver como la función para obtener el total de una venta, se delega a la línea de venta, que se compone de múltiples productos, y ésta delega la obtención del precio al producto para obtener el total.

De una forma similar a la del ejemplo, en el proyecto se utiliza este patrón para implementar los efectos del menú de combate, ya que los eventos detectados sobre los botones de navegación son delegados al listado de ítems del menú, mientras que éste envía la información al objeto que se ve afectado para generar el efecto sobre él.

5.4 Diseño estético y de interfaz

Uno de los objetivos de este proyecto era mantener la estética original del dispositivo, para ello se han usado los sprites originales del juego para todo elemento visual relativo al combate. También se han evitado recursos externos como imágenes de fondo para mantener la estética original.

5.4.1 Diseño artístico

Los sprites de los distintos personajes se han obtenido de la Digimon Wiki [22][23][24] en el apartado de Galería, donde se recrean los sprites originales del dispositivo. Para los sprites de los menús se han utilizado los sprites de un proyecto en GitHub [25] publicado por el usuario Kaisadilla sobre este mismo dispositivo.

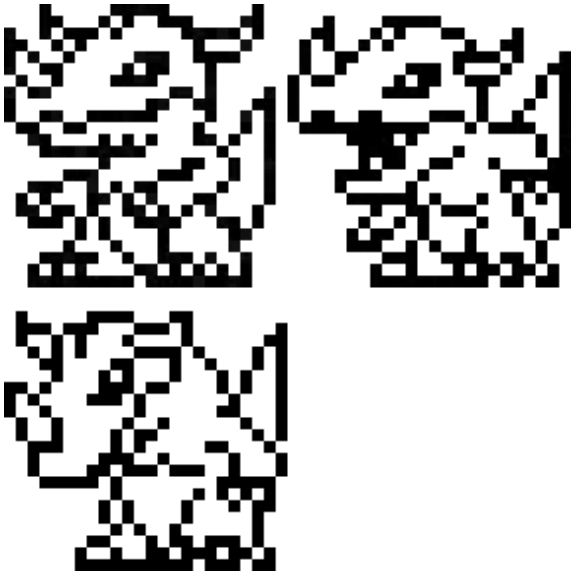

Ilustración 21: Sprite múltiple utilizado para representar al personaje aliado y enemigo intermedio

Por último, para textos dinámicos y personalizados, se ha utilizado la fuente de texto gratuita Pixel Digivolve [26] de Pixel Sagas, la cual, pese a no ser idéntica, se inspira en las fuentes utilizadas para estos dispositivos.

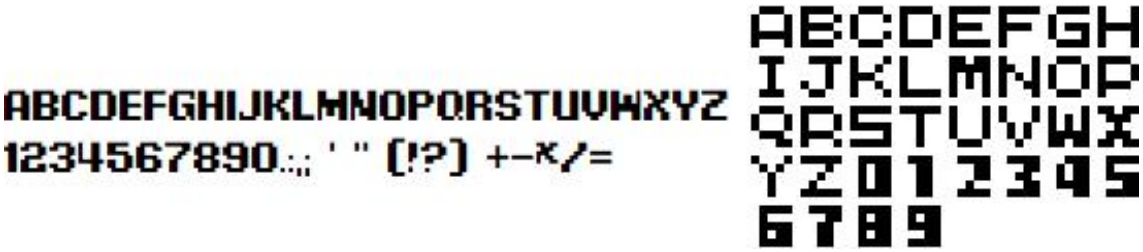

Ilustración 22: Comparativa entre la fuente Pixel Digivolve (izq.) frente a la fuente original del dispositivo (der.)

Existen ciertas diferencias en las fuentes de texto. En la aplicación se puede apreciar la diferencia dado que los iconos del menú de combate utilizados incluían el texto original, mientras que el texto para mostrar la vida de los personajes y el usado en los botones del menú de inicio de la aplicación se crearon utilizando la fuente Pixel Digivolve.

La razón para usar dicha fuente es para evitar un exceso de trabajo manipulando los sprites de cada carácter para formar las cadenas de texto correspondientes y sobre todo, para simplificar el proceso de actualizar la vida de los personajes, ya que trabajar con sprites que representaran la vida resultaría bastante complicado. A pesar de esto, fue una idea que se consideró y se trabajó, pero finalmente quedó descartada.

Por último comentar, que dado que el menú de inicio no es parte del dispositivo original, sino una herramienta utilizada para simplificar el proyecto y hacerlo funcional, se ha optado por usar una imagen de fondo e incluir el logotipo original de la serie, el cual también fue utilizado en el embalaje del dispositivo que recrea. Sin embargo, la estética retro se mantiene utilizando la fuente Pixel Digivolve en los botones de dificultad.

5.4.2 Diseño de interfaz

La aplicación contiene dos interfaces de usuario, la del menú principal, que permite escoger la dificultad del combate, y la del menú de acción en el combate.

La interfaz del menú principal no sigue ninguna referencia, ya que no existía en el videojuego original, por lo que busca ser sencilla y fácil de usar mientras mantiene la estética retro original.

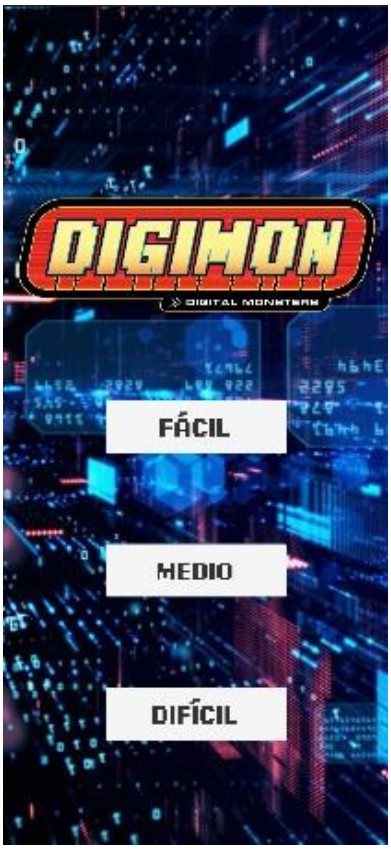

Ilustración 23: Interfaz del menú principal

Como se puede ver en la imagen, los botones son lo suficientemente grandes para destacar con el fondo blanco sobre la imagen usada, de forma que resulte fácil seleccionar la opción sin problema, y dado que no hay demasiadas opciones, no es necesario implementar ningún tipo de navegación en el menú a través de deslizamiento o botones de dirección.

En el menú de combate se ha tenido de referencia la idea original, tratando de mantener el concepto de mostrar una única opción en pantalla y desplazándose en el menú a través de botones.

Dado que los dispositivos móviles no suelen contar con botones físicos frontales como el dispositivo original que adapta, se ha decidido desplazar las flechas que incluían los distintos iconos del menú en los laterales, ubicándolas en la parte inferior de la pantalla para aprovechar mejor las dimensiones de la pantalla, permitiendo hacer los iconos más visibles, y separando sus cajas de interacción. De esta forma se evitan posibles errores en la selección de la acción, ya que una vez seleccionada la opción de ese turno, no se puede deshacer el cambio.

Ilustración 24: Interfaz del menú de combate mostrando los tamaños del botón mediante un rectángulo blanco

La anterior imagen muestra la interfaz desde el editor Unity, por lo que no representa la visión final del menú de la aplicación. Algunos elementos, como las flechas amarillas y naranjas, o el círculo azul y la cruz blanca del centro son formas de representar información para el desarrollador.

La razón por la que se ha hecho la captura desde esta vista es para poder mostrar fielmente el tamaño real de los distintos botones de la interfaz, ya que el objetivo final es que las cajas de interacción no sean visibles, y simplemente se use la el sprite como referencia del tamaño.

Nótese que la distancia entre los distintos botones es suficientemente grande para evitar pulsar en un botón equivocado, y las cajas de interacción son del tamaño suficiente para poder pulsar los distintos botones sin dificultad, y al mismo tiempo evitar pulsarlos accidentalmente si no se pretende hacerlo.

Capítulo 6: Implementación

6.1 Introducción

A lo largo de este capítulo se describirán las distintas tecnologías utilizadas para el desarrollo del proyecto, comenzando por las distintas aplicaciones y plataformas de apoyo, y continuando por la explicación y desarrollo de los algoritmos y procesos de la plataforma de desarrollo de la aplicación.

6.2 Herramientas externas a la aplicación

Para el desarrollo del proyecto, se han usado múltiples plataformas y herramientas de apoyo, ya sea para desarrollar esta documentación o para generar recursos útiles para el desarrollo y mantenimiento de la aplicación. A continuación expondré estas distintas plataformas, describiéndolas y comentando el uso realizado de ellas en este proyecto.

6.2.1 GitHub

GitHub [27] es la plataforma online de gestión de proyectos y control de versiones que más se popularizó de entre todas aquellas que utilizan el software Git. Permite alojar proyectos en la nube de manera pública y privada. El mayor valor de esta plataforma es el control que ofrece para gestionar proyectos en conjunto al permitir subir diferentes versiones del código del proyecto, y generar ramas de versiones en las que generar cambios sobre funcionalidades específicas del proyecto, permitiendo organizar el código y encapsular errores para mejorar el trabajo colectivo.

Ilustración 25: Logotipo de la plataforma GitHub

GitHub fue creada a finales de 2007 y desarrollada por Chris Wanstrath, P. J. Hyett, Tom Preston-Werner y Scott Chacon. En 2018 fue comprado por Microsoft, lo que provocó cierta preocupación sobre los usuarios por posibles cambios en la política de la plataforma, pero se ha mantenido siendo la plataforma más importante de colaboración para proyectos de código abierto.

El uso de la plataforma en este proyecto ha sido bastante reducido, utilizándose únicamente como herramienta para generar copias de seguridad del proyecto, acceder a ciertos recursos gráficos del proyecto de Kaisadilla [25], utilizados para evitar la necesidad de generar estos recursos de manera manual.

6.2.2 Astah

Astah [28] es un software de modelado UML que permite generar mapas mentales y múltiples diagramas de diseño y arquitectura software con soporte para varios lenguajes de programación para facilitar aún más el uso de la herramienta en la creación de diagramas.

Ilustración 26: Logotipo de la herramienta Astah

Astah surge de un cambio de nombre de la herramienta realizado en 2009. Anteriormente era conocida como Jude, y fue creada por Kenji Hiranabe y vendida en el mercado japonés por el año 2004. Actualmente es una herramienta de pago conocida a nivel mundial, que tiene múltiples versiones del software con algunas características específicas y que hemos utilizado a lo largo del grado.

En concreto la versión utilizada para el proyecto ha sido Astah Professional, cuya licencia ha sido proporcionada por la universidad, y ha sido utilizado para realizar todos los diagramas del proyecto mostrados en la documentación.

6.2.3 Visual Studio

Microsoft Visual Studio [29] es un IDE (entorno de desarrollo integrado), o lo que es lo mismo, una herramienta de soporte para el desarrollo de código en múltiples lenguajes. Visual Studio tiene soporte para gran cantidad de lenguajes tales como C++, C#, Java, Phyton o .NET, además de la posibilidad de integrarse con otras plataformas de desarrollo como la utilizada en este proyecto (Unity), para ayudar en el autocompletado de funciones y variables preestablecidas del motor externo.

Ilustración 27: Logo de Visual Studio

Visual Studio fue originalmente desarrollado en 1997 por Microsoft, pero sufre grandes cambios en 2005, cuando se comienzan a lanzar de manera gratuita versiones básicas de la aplicación para distintas plataformas y en 2013 al añadir más versiones de la aplicación.

Visual Studio se ha utilizado para la creación de los distintos scripts de código en lenguaje C# utilizados en el programa a lo largo del proyecto, ya que Unity permite la integración de ambas plataformas de desarrollo, ofreciendo ayudas en la identificación de errores y compilación de código. La versión utilizada en el proyecto ha sido Visual Studio 2019 ya que disponía previamente de ella y no he tenido la necesidad de actualizar a nuevas versiones para obtener funcionalidades que nuevas versiones pudieran ofrecer.

6.2.4 Otras herramientas

Además de las herramientas previamente mencionadas, se han hecho uso de otras herramientas que no están relacionadas con la programación o el diseño del software, pero que han sido muy importantes para el desarrollo del proyecto. A continuación iré enumerándolas y describiendo ligeramente su uso.

- Adobe Photoshop: Un programa de edición y diseño gráfico, utilizado en el proyecto para modificar, combinar y ajustar sprites. Combinar los sprites en una sola imagen, permite utilizar las animaciones de Unity que se han utilizado. Modificar los sprites del menú de combate ha permitido ajustar la interfaz de usuario, al eliminar las flechas de los sprites originales, y generando imágenes independientes para éstas. Por último, también permite ajustar el tamaño de las imágenes de una manera más profesional y precisa que utilizando la opción de “escala” de Unity.
- Google Drive: Plataforma de almacenamiento de archivos en la nube. Algunos de los recursos gráficos fueron obtenidos de esta plataforma, pero están siendo sustituidos para evitar disparidad en la estética.
- Microsoft Project: Aplicación de gestión de proyectos, se usó para hacer una planificación inicial de las fases del proyecto, pero no se hizo un gran uso de ella debido a las dificultades de disponibilidad en el horario, lo que hizo que la planificación inicial no se ajustara con el estado del proyecto.

6.3 Plataforma de desarrollo

Unity [30] es un motor de videojuegos para desarrollar aplicaciones sobre múltiples plataformas tales como PlayStation, Xbox, Android, iOS, Windows, Linux, Mac o Nintendo Switch, además de navegadores web y diversos sistemas de TV inteligente.

Unity funciona como plataforma de desarrollo para sistemas Windows, Linux y Mac, y es junto con Visual Studio el IDE utilizado para sacar el proyecto adelante. El motor contiene una gran cantidad de características que le han hecho muy popular. A lo largo de la sección describiré las características principales utilizadas en el desarrollo del proyecto, haciendo especial hincapié en aquellas que más me han llamado la atención y que más he tenido que investigar para el avance de la aplicación.

Unity incluye un editor gráfico con el que se pueden realizar cambios visuales sobre el proyecto, pero las funcionalidades más avanzadas deben realizarse mediante código externo al editor en forma de scripts, que se acoplan a los objetos. Estos scripts están programados en lenguaje C# de manera predeterminada, pero es posible utilizar otros lenguajes compatibles con .NET siempre que puedan compilar una biblioteca de enlace dinámico (DLL) al añadir el archivo .dll en el proyecto. El motor gráfico utiliza OpenGL y Direct3D, siendo esta última la opción estándar para Windows.

6.3.1 El uso de Prefabs

Una de las funcionalidades integradas en el editor de Unity es el uso de los denominados Prefabs, los cuales son GameObject guardados en el proyecto como un Asset más, lo que facilita la reutilización de estos GameObject al permitir editar las características sobre todas las instancias del GameObject al mismo tiempo. Por otro lado, también permite realizar la operación opuesta, para lo cual se ha utilizado en este proyecto. En el proyecto el uso de Prefabs permite cargar al enemigo de manera dinámica dependiendo de la dificultad. La aplicación comprueba la dificultad seleccionada por el usuario y genera una instancia del Prefab correspondiente.

Además de permitir la carga dinámica de los enemigos, el uso de Prefabs permite la modificación de estadísticas como la vida actual sobre la instancia generada a través del Prefab sin modificar la estadística del Prefab original. De esta forma conseguimos que al reiniciar el combate se genere una nueva instancia del Prefab con las estadísticas iniciales, sin tener que restablecer las del combate anterior, simplemente destruyendo la escena cuando el combate acaba al volver al menú de inicio.

6.3.2 El uso de DontDestroyOnLoad

En Unity el cambio de una escena a otra provoca la destrucción de los objetos que existían en ésta. A veces se quiere mantener cierta información entre escenas que es demasiado compleja para guardarla mediante variables static en clases static o simplemente, es más conveniente que la información quede ligada al objeto relacionado. Para estos casos existe una función en Unity que permite evitar la destrucción de un objeto; de esta forma, DontDestroyOnLoad permite evitar la pérdida de información entre escenas, pero a la vez añade la pequeña contraparte de poder generarse duplicados al volver a la escena donde el objeto con esta función fue creado originalmente.

En este proyecto se utiliza esta función para el controlador que maneja los estados y variables del combate, y para los personajes participantes en el combate, de manera que se guarde la información relativa a la vida actual de éstos en cada turno. Para evitar el problema de los duplicados se utilizó el patrón Singleton sobre estos controladores, evitado su duplicación.

Los personajes del combate son instanciados a través del controlador en la fase inicial de éste, ya que deben instanciarse de manera dinámica dependiendo de la dificultad seleccionada por el usuario. Esto evita la duplicación de estos objetos, ya que la fase inicial del combate no vuelve a ejecutarse hasta que éste no termina y se genera un combate nuevo.

6.3.3 El uso de corrutinas

En este proyecto muchos de los eventos generados por el usuario desencadenan una secuencia de cambios visuales guiada por el tiempo, en la que el usuario no interviene, pero debe estar controlada para evitar el solapamiento de animaciones. Cuando se generan eventos a lo largo del tiempo, es necesario el uso de corrutinas, dado que realizar estos procesos sobre el hilo principal generando bloqueos a través de la comúnmente utilizada función Sleep() o funciones similares, provocará el congelamiento de la aplicación, inhabilitando los cambios gráficos que se pretenden mostrar. Esto es debido a que el sistema queda bloqueado hasta que termina la operación, lo que impide que se actualice la interfaz gráfica del juego.

En lenguaje C#, una corrutina se corresponde con una función con un tipo de retorno IEnumerator a la cual se accede mediante la función StartCoroutine() identificando el nombre de la corrutina a ejecutar dentro de los paréntesis.

Las corrutinas permiten devolver el control a Unity sin terminar la ejecución de la función al completo, añadiendo sentencias yield a lo largo de la función. De esta forma, a través de sentencias como “yield return new WaitForSecondsRealtime()” especificando el tiempo de retardo que se quiere añadir a la función dentro del paréntesis en forma de segundos y en formato float.

Las corrutinas también se pueden anidar añadiendo sentencias como “yield return StartCoroutine()”. Esto permite establecer una secuencia de corrutinas independientes que se deben ejecutar de manera consecutiva sin tener que realizar las llamadas de manera independiente desde el hilo principal. En la aplicación se utilizan las anidaciones en corrutinas como la que muestra y actualiza la vida, ya que dicha corrutina no solo es llamada al finalizar un ataque, sino también tras realizar una curación sobre el aliado, por lo que tiene sentido que las llamadas a esta corrutina se realicen directamente sobre otras, y no sobre el hilo principal, lo que obligaría a incluir más condicionales.

El control de estos eventos temporales se realiza a través de la función de eventos Update, predefinida por Unity, la cual se ejecuta una vez por frame. Para evitar el solapamiento de eventos en el combate, se utilizan condicionales en relación al estado de éste, identificado como una variable tipo enum, y que funciona como bandera para restringir los eventos en la función Update, de manera que según el estado en el que se encuentre el combate en el momento de ejecución de la función, solo se podrán ejecutar ciertas corrutinas.

6.3.4 Las funciones de evento

En el apartado anterior comento el uso de la función de eventos Update. Unity tiene una serie de funciones predeterminada para todo script que se añade a un proyecto. Estas funciones no necesitan estar definidas si no se van a usar, pero en este proyecto se utilizan varias de estas funciones, para lo cual ha sido necesario comprenderlas y conocer cuándo y en qué orden son ejecutadas. A continuación mostraré un gráfico obtenido del manual de Unity [31] en el que se muestra la secuencia de procesos realizada sobre los scripts que se agregan a un GameObject en Unity.

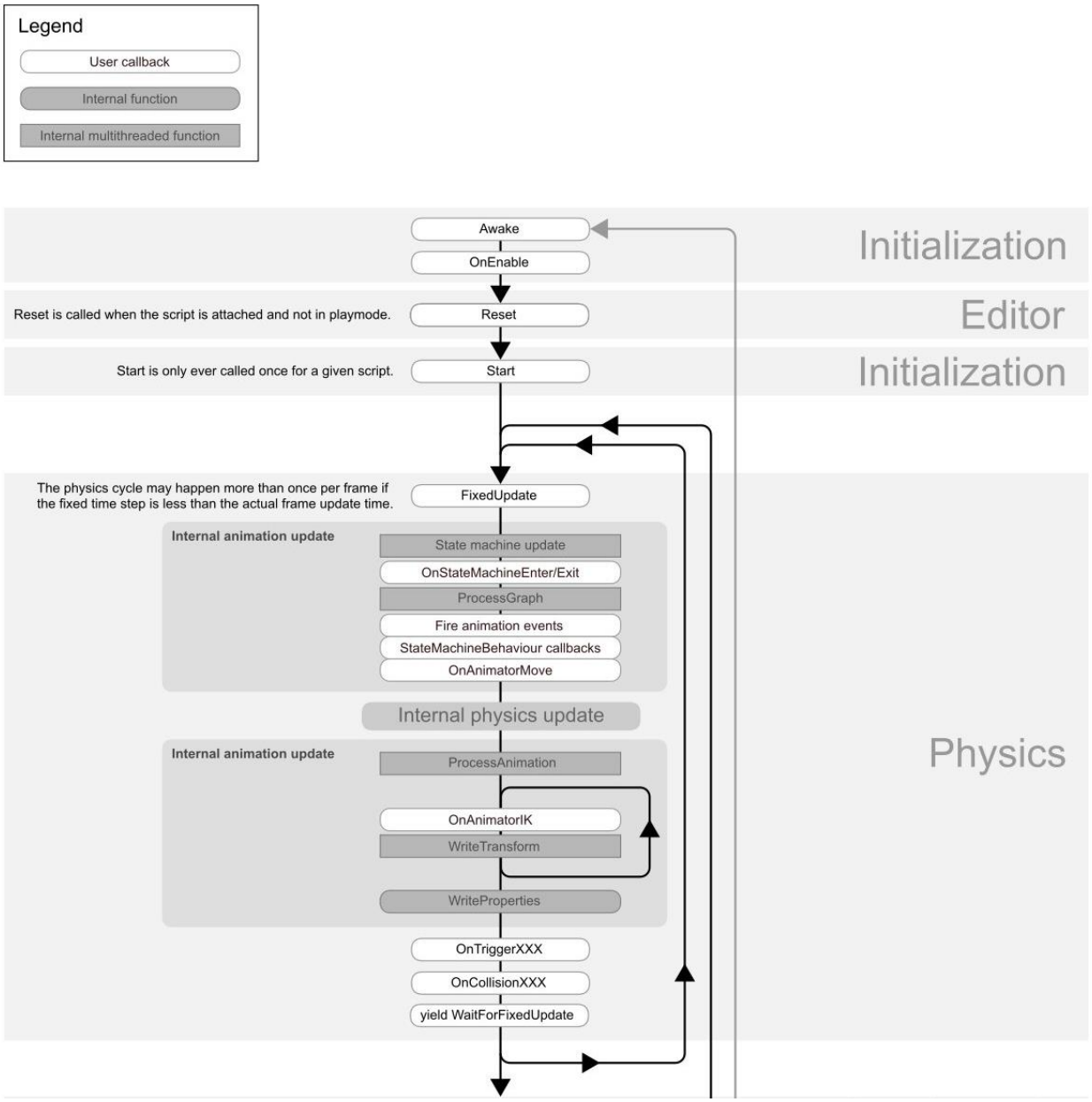

Ilustración 28: Diagrama de secuencia de procesos en los scripts (pt. 1)

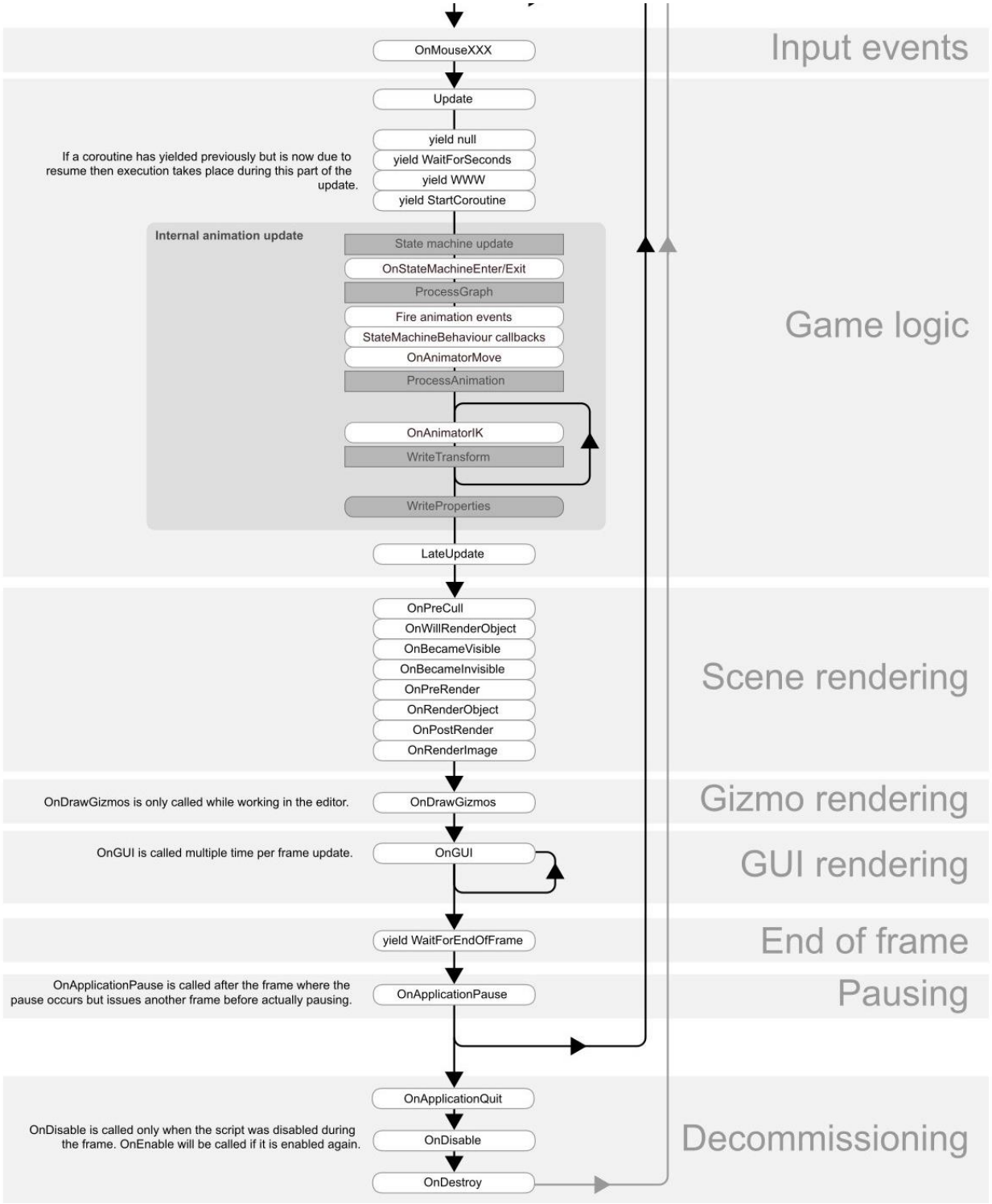

Ilustración 29: Diagrama de secuencia de procesos en los scripts (pt. 2)

Del diagrama mostrado, nos fijaremos en las denominadas “User callback” de la leyenda y en la sección a la que pertenecen.

En la fase Initialization del objeto se ejecutan las funciones Awake, OnEnable y Start. Ambas se usan en este proyecto, y la principal diferencia es que la función OnEnable se ejecuta siempre que un objeto instanciado pasa a estar activo, mientras que Awake solo se ejecuta durante la instanciación inicial, y no se vuelve a ejecutar sobre el objeto. Esto es útil para evitar la destrucción y reinstanciación de objetos cada vez que se quieren ocultar éstos. De esta forma solo es necesario cambiar su estado de activo a inactivo o viceversa.

Las funciones Awake y Start son muy similares, y simplemente permiten dividir la inicialización en varias fases. Otra pequeña diferencia es que la función Start no será llamada si durante la inicialización del objeto el script es deshabilitado.

Durante la fase Physics y Game Logic se pueden ver varias funciones relacionadas con el componente Animator de Unity. No detallaré sobre las distintas funciones mostradas dado que el uso dado a este componente en el proyecto es bastante simple, pero es interesante ver que el componente funciona a través de una máquina de estados. También puede verse diversas llamadas yield y la función Update.

La función Update se ejecuta una vez por frame y se utiliza para comprobar cambios en variables u objetos en el juego. Las llamadas yield, por otro lado son las usadas en corrutinas para suspender su ejecución. La ejecución es devuelta tras ejecutarse todas las funciones Update de los objetos activos, cumpliéndose la regla que se añada a la llamada yield. Es decir, si la función es yield WaitForSeconds, la corrutina suspenderá su ejecución durante el número de segundos que se establezca, y pasado ese tiempo, esperará a que se ejecuten todas las funciones Update del frame para volver a ejecutarse.

Por último, detallaré la fase “Decommissioning”, en la que encontramos las funciones OnDisable y OnDestroy. OnDisable y OnDestroy funcionan de manera similar a las mencionadas anteriormente OnEnable y Start.

La función OnDisable es llamada cuando un objeto activo cambia de estado a inactivo, al contrario que OnEnable, mientras que OnDestroy, en contraposición a Start, que se ejecuta una única vez al instancializarse el objeto, se ejecuta una única vez, pero cuando el objeto va a ser destruido, ya sea de manera manual o a través de un cambio de escena.

6.3.5 El uso de Animator

Para generar los cambios de animación en los scripts se ha utilizado el componente Animator. Este componente permite generar animaciones cambiando el sprite a mostrar cuando se trabaja con un sprite múltiple, además de otros parámetros como la posición. Primero se asigna el componente Animator al sprite múltiple, y se añade una animación al Animator, que funciona como controlador, a continuación se configura la animación.

La configuración se realiza mediante el editor de Unity asignando las variables como la posición y el sprite individual a mostrar, esto se realiza a través de una barra de tiempo, que indica la duración de la animación.

Cuando hay varias animaciones asignadas a un controlador Animator, es necesario asignar qué animaciones se realizarán según qué situación. Para ello, el editor muestra una máquina de estados en la que se incluyen las distintas animaciones asignadas a ese controlador, pudiendo transicionarse entre las distintas animaciones asignando un tiempo de duración a la transición o a través de condiciones asignadas a parámetros. Estos parámetros pueden ser valores boolean o simples trigger.

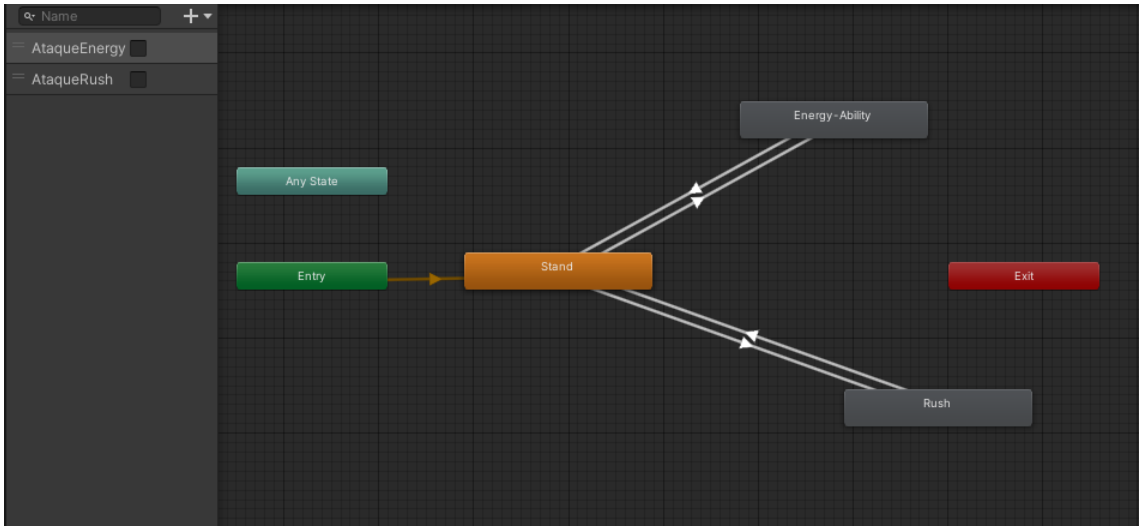

Ilustración 30: Máquina de estados con las animaciones de uno de los objetos del proyecto

En la imagen se puede ver cómo se muestra la máquina de estados de uno de los objetos del proyecto. A la izquierda se pueden ver los distintos parámetros que tiene el controlador, que son utilizados para gestionar la transición entre animaciones. En este caso son parámetros boolean, y las transiciones se gestionan mediante el valor que tengan los parámetros.

6.4 Algoritmos considerados

En el proyecto se consideraron dos algoritmos para gestionar el procesamiento de ataques del aliado a lo largo del combate, un algoritmo basado en la sustracción de fondo de imagen, y otro basado en la intensidad de color detectado por la cámara en cada momento. A pesar de que en la implementación final, suponiendo que no haya habido cambios de última hora, no se han llegado a implementar ninguno de estos algoritmos completamente, creo que es importante nombrarles y explicar su funcionamiento, pues es uno de los trabajos a futuro sobre los que hablaré más adelante.

- Algoritmo basado en sustracción de fondo: Se toman 2 imágenes de frames consecutivos y un valor límite que determine qué puede considerarse movimiento y qué no puede considerarse movimiento. A continuación se calcula la diferencia entre las imágenes considerando el valor límite para obtener una nueva imagen. Si el valor de la diferencia es menor al valor límite se le asigna el color negro, y para aquellos pixeles con valor mayor al límite, se le asigna el color blanco. Después se asignaría un porcentaje de pixeles para considerar si ha habido movimiento o no.
- Algoritmo basado en la intensidad del color: La idea de este algoritmo se centra en la idea de que la imagen de la cámara pasará por tres estados antes de llegar a considerar que ha habido movimiento. Un estado en el que la cámara detectará una variedad de valores cromáticos, un estado en el que quedará tapada prácticamente en su totalidad durante varios frames, lo que significa que los valores de color obtenidos en los pixeles de la cámara serán de un color muy oscuro y la variedad cromática disminuirá, y un último estado en el que la variedad cromática volvería a ser similar a la del estado inicial.

De los dos algoritmos mencionados, el primero resulta complicado de implementar, debido a que implica que se realicen muchos cálculos por frame, y además podría generar problemas en lugares con mucha variedad de colores, de manera que un mínimo movimiento del foco de la cámara generara esa detección de movimiento.

El segundo algoritmo tiene más sentido, y es el que se ha usado para conseguir la implementación, aunque tiene la desventaja de no distinguir bajo entornos de muy baja luminosidad. Probablemente sería necesario el uso de algoritmos más complejos y de un tratamiento de las imágenes avanzado del que no dispongo ahora mismo.

Capítulo 7: Pruebas

7.1 Introducción

A lo largo del desarrollo del proyecto, se han realizado múltiples pruebas y testeos de la aplicación, en este capítulo mostraré las distintas pruebas realizadas para comprobar el correcto funcionamiento de la aplicación, tanto desde el dispositivo de desarrollo, en cuyo caso serán pruebas de módulos parciales, como desde el dispositivo final del producto.

7.2 Pruebas desde el dispositivo de desarrollo

A continuación se listan las distintas pruebas realizadas durante el proceso de desarrollo de la aplicación, ordenadas de manera cronológica.

Durante estas pruebas la aplicación no está completa, por lo que son pruebas para comprobar la implementación de módulos aislados (escenas) y de la unión entre los distintos módulos.

Prueba 1	**Cambio de escena desde el menú principal**
Resultados esperados	Al pulsar en uno de los botones del menú principal se cambia a la escena de combate
Resultados obtenidos	Resultados esperados

Tabla 20: Prueba 1-Cambio de escenas desde el menú principal

Prueba 2	**Carga del menú de combate**
Resultados esperados	Al cargarse la escena de combate por primera vez se muestra al personaje aliado y enemigo y la escena pasa al menú de combate
Resultados obtenidos	Resultados esperados

Tabla 21: Prueba 2-Carga del menú de combate

Prueba 3	**Interactividad del menú de combate**
Resultados esperados	Al pulsar los botones de navegación del menú se actualiza el botón de acción mostrado desplazándose desde el lado del botón pulsado
Resultados obtenidos	Resultados esperados

Tabla 22: Prueba 3-Interactividad del menú de combate

Prueba 4	**Selección de la acción de atacar**
Resultados esperados	Se carga la escena de detección de la cámara
Resultados obtenidos	Resultados esperados

Tabla 23: Prueba 4-Selección de la acción de atacar

Prueba 5	**Detección de ataque con la cámara**
Resultados esperados	Se realiza la detección del ataque y se carga la escena de combate
Resultados obtenidos	La escena de la cámara queda bloqueada porque el dispositivo detecta una cámara virtual, generando un error en el programa al intentar obtener las imágenes de ésta

Tabla 24: Prueba 5-Detección de ataque con la cámara

Se ha ajustado el código para evitar la sección que involucra a la cámara y evitar el error.

Prueba 6	**Detección de ataque con la cámara**
Resultados esperados	Se realiza la detección del ataque y se carga la escena de combate
Resultados obtenidos	Resultados esperados

Tabla 25: Prueba 6-Detección de ataque con la cámara

Prueba 7	**Procesamiento de turno de ataque sin victoria o derrota**
Resultados esperados	Se generan las animaciones de ataque y se vuelve al menú de combate
Resultados obtenidos	La escena no muestra a los personajes y genera un error al intentar referenciar a los objetos relativos a los personajes

Tabla 26: Prueba 7-Procesamiento de turno de ataque sin victoria o derrota

Se ha añadido la función DontDestroyOnLoad en los personajes para evitar la pérdida de los objetos y la información relativa a éstos.

Prueba 8	**Procesamiento de turno de ataque sin victoria o derrota**
Resultados esperados	Se generan las animaciones de ataque y se vuelve al menú de combate
Resultados obtenidos	Tras generarse las animaciones el programa queda bloqueado en la escena de combate

Tabla 27: Prueba 8-Procesamiento de turno de ataque sin victoria o derrota

El problema venía generado de un bucle producido en la función Update, bloqueando el proceso y con ello la escena. Se ha rediseñado el control de estados del combate para evitar los bucles en esta función.

Prueba 9	**Procesamiento de turno de ataque sin victoria o derrota**
Resultados esperados	Se generan las animaciones de ataque y se vuelve al menú de combate
Resultados obtenidos	Resultados esperados

Tabla 28: Prueba 9-Procesamiento de turno de ataque sin victoria o derrota

Prueba 10	**Actualización de la vida de los personajes**
Resultados esperados	La vida de los personajes se actualiza correctamente calculando el daño recibido en el turno y se mantiene entre turnos
Resultados obtenidos	Resultados esperados

Tabla 29: Prueba 10-Actualización de la vida de los personajes

Prueba 11	**Procesamiento de turno de ataque con victoria o derrota**
Resultados esperados	Se generan las animaciones de ataque y se vuelve al menú de inicio
Resultados obtenidos	Resultados esperados

Tabla 30: Prueba 11-Procesamiento de turno de ataque con victoria o derrota

Prueba 12	**Selección de la acción de curar aliado por primera vez en un combate**
Resultados esperados	Se muestra la escena de combate donde se actualiza la vida del aliado sin superar su vida inicial
Resultados obtenidos	Resultados esperados

Tabla 31: Prueba 12-Selección de la acción de curar aliado por primera vez en un combate

Prueba 13	**Selección de la acción de curar aliado tras la primera vez en un combate**
Resultados esperados	El botón de acción se muestra inactivo, impidiendo su uso
Resultados obtenidos	Resultados esperados

Tabla 32: Prueba 13-Selección de la acción de curar aliado tras la primera vez en un combate

Prueba 14	**Selección de la acción de abandonar combate**
Resultados esperados	Se carga la escena de menú inicial y se reestablece la vida de los personajes y la acción de curación
Resultados obtenidos	Resultados esperados

Tabla 33: Prueba 14-Selección de la acción de abandonar combate

Prueba 15	**Selección de dificultad desde el menú de inicio**
Resultados esperados	Se muestra la escena de combate con el enemigo correspondiente a la dificultad seleccionada
Resultados obtenidos	Resultados esperados

Tabla 34: Prueba 15-Selección de dificultad desde el menú de inicio

7.3 Pruebas desde el dispositivo final

En estas pruebas el desarrollo de la aplicación ya está completo por lo que son pruebas de caja negra en las que se comprueba la correcta funcionalidad de la aplicación en conjunto desde el entorno intencionado.

Prueba 16	**Selección de dificultad de combate**
Resultados esperados	Se carga el combate con el enemigo correspondiente a la dificultad seleccionada y se muestra el menú de combate
Resultados obtenidos	Resultados esperados

Tabla 35: Prueba 16-Selección de dificultad de combate

Prueba 17	**Selección de la acción de atacar**
Resultados esperados	Se carga la escena de detección de la cámara
Resultados obtenidos	Resultados esperados

Tabla 36: Prueba 17-Selección de la acción de atacar

Prueba 18	**Detección de ataque con la cámara**
Resultados esperados	Se realiza la detección del ataque y se carga la escena de combate
Resultados obtenidos	Se realizan detecciones no intencionadas durante el ataque

Tabla 37: Prueba 18-Detección de ataque con la cámara

Se ha reajustado los valores del algoritmo para evitar detecciones de movimiento en entornos con luz moderada o colores oscuros. Por el funcionamiento del algoritmo, podrían seguir ocasionándose detecciones no intencionadas en entornos con luz muy reducida, pero no deberían ocurrir bajo entornos normales.

Prueba 19	**Detección de ataque con la cámara**
Resultados esperados	Se realiza la detección del ataque y se carga la escena de combate
Resultados obtenidos	Resultados esperados

Tabla 38: Prueba 19-Detección de ataque con la cámara

Prueba 20	**Selección de la acción de curar aliado por primera vez en un combate**
Resultados esperados	Se muestra la escena de combate donde se actualiza la vida del aliado sin superar su vida inicial
Resultados obtenidos	Resultados esperados

Tabla 39: Prueba 20-Selección de la acción de curar aliado por primera vez en un combate

Prueba 21	**Selección de la acción de curar aliado tras la primera vez en un combate**
Resultados esperados	El botón de acción se muestra inactivo, impidiendo su uso
Resultados obtenidos	Resultados esperados

Tabla 40: Prueba 21-Selección de la acción de curar aliado tras la primera vez en un combate

Prueba 22	**Selección de la acción de abandonar combate**
Resultados esperados	Se carga la escena de menú inicial y se reestablece la vida de los personajes y la acción de curación
Resultados obtenidos	Resultados esperados

Tabla 41: Prueba 22-Selección de la acción de abandonar combate

La mayoría de estas pruebas ya habían sido realizadas en el dispositivo de desarrollo, pero es necesario comprobar que el cambio de sistema no afecta a la aplicación.

Capítulo 8: Conclusiones y trabajo a futuro

8.1 Declaraciones finales sobre el proyecto

Los objetivos de este proyecto eran aprender a manejar nuevas plataformas de desarrollo software para videojuegos intentando hacer una recreación fiel del sistema de combate del dispositivo D-Tector original. Estos objetivos se han cumplido, no en su totalidad, ya que se han realizado adaptaciones del sistema original para adaptar el proyecto, pero sí en gran parte.

Se ha aprendido a desarrollar aplicaciones móviles mediante Unity, y a manejar sprites 2D con animaciones. También se han aprendido varias técnicas de la plataforma de desarrollo para mantener la estabilidad en los píxeles tras su escalado.

Se ha conseguido realizar una adaptación del sistema de combate original, utilizando la cámara para simular la interacción del dispositivo con su sensor.

Por otro lado, como bien he dicho, no he conseguido realizar una adaptación completa del sistema de combate, empezando por la falta de algunas de las animaciones del juego original, además de otras funciones como la de añadir múltiples personajes para aliado, que puedan cambiarse en mitad del combate. El valor estratégico que se pretendía dar al juego, se ha reducido en cálculos de estadísticas de los personajes, aunque no se ha llegado a implementar los patrones de ataque específicos de cada enemigo.
Nota: La asignación de patrones de ataque a los enemigos se ha conseguido implementar a última hora, añadiendo probabilidades a los distintos tipos de ataque según el enemigo, en la guía de usuario se declaran dichas probabilidades.

8.2 Trabajo futuro

La aplicación de este proyecto, pese a ser funcional y cumplir con el objetivo de entretener, tiene muchas mejoras posibles. A continuación detallaré aquellas en las que he ido pensando a lo largo del proyecto, y que no han podido llegar a desarrollarse.

8.2.1 Mejoras y cambios de funcionalidad existentes

Como bien he comentado, algunas de las funcionales existentes de la aplicación son mejorables. Estos aspectos serían los primeros en los que pondría mi atención en futuras versiones de la aplicación, empezando por la implementación de animaciones y

efectos de sonido faltantes en la versión actual, y la modificación de algunas de las animaciones existentes para que coincidan con las del dispositivo original.

Es importante entender que, dado que el juego ha sido diseñado para ejecutarse bajo una resolución específica, existe la posibilidad de que en dispositivos con resoluciones muy dispares, la interfaz de usuario resulte difícilmente accesible y los sprites se muestren de manera no intencionada. Este aspecto es algo importante a corregir en el futuro si se continúa desarrollando este proyecto.

Otro punto interesante sería la mejora del algoritmo de detección de ataque mediante la cámara, para evitar inconsistencias en entornos muy oscuros, sin afectar al rendimiento en otros entornos.

También es verdad que el objetivo de añadir patrones de ataque propios a cada enemigo para añadir cierto valor estratégico al juego quedó en el olvido relegándose a un pequeño estudio numérico de las estadísticas existentes para añadir dificultad. Esto permitiría añadir más dificultad al juego y le haría más interesante.

De esta sección, los aspectos previamente mencionados serían la principal prioridad, pero existen varios aspectos pensados que me han resultado interesantes para realizar. Principalmente son cambios en la funcionalidad a partir de opciones extra para el usuario.

Aunque no es muy usual, existe la posibilidad de que algunos dispositivos móviles no contengan cámara, esto impide el uso del videojuego actualmente, por lo que se ha planteado añadir una opción para seleccionar el tipo de ataque a realizar sin utilizar la cámara para que aquellos dispositivos pudieran jugar sin problema, a pesar de que es la principal característica del juego.

Por último, se ha considerado el uso de sprites con color para modernizar el juego. Para ello se ha pensado en añadir una opción al usuario para cambiar entre la interfaz a color y la interfaz original, pero esto sería un cambio planteado para realizar después del resto de funcionalidades, ya que la intención del proyecto es adaptar el dispositivo original.

8.2.2 Implementación de funcionalidades de combate extra

El dispositivo original tenía algunas funcionalidades que no se han implementado para simplificar el proyecto. El principal cambio que me gustaría implementar de esta sección sería la posibilidad de cambiar al personaje aliado en mitad del combate por otro, de manera que tengan una cantidad de vida independiente entre ellos.

Otro cambio sería la implementación de transformaciones de los personajes llamadas “digievoluciones”, esto sería similar a un cambio de personaje, pero a diferencia del cambio de personaje, no podría volverse a utilizar al anterior personaje tras la transformación. Esta implementación podría utilizarse tanto en aliados como en enemigos, haciendo posible combates con múltiples fases que, junto con la implementación previamente mencionada de patrones de ataque, podría dar un gran dinamismo al juego.

8.2.3 Implementación de otras funcionalidades externas al combate

Como se comenta al comienzo del proyecto, los dispositivos originales disponían de un contador de pasos, y los combates se generaban aleatoriamente al avanzar, esta implementación se pensó originalmente, pero se evitó ya que consideraba que la extensión del proyecto sería demasiado grande si se incluía, por lo que se simplificó con la implementación de un menú y niveles de dificultad.

Para la implementación de esta funcionalidad, se planteó principalmente el uso del giroscopio de los dispositivos móviles, pues en el dispositivo original era muy común agitar el dispositivo para simular los pasos del usuario, y así evitar andar por la calle con el dispositivo, ya que terminaba siendo algo bastante tedioso para avanzar en el juego. Utilizando el giroscopio se podría recrear el efecto original del dispositivo, de manera que se contaran las sacudidas y se generara una probabilidad de iniciar un combate contra un enemigo aleatorio con cada cierta cantidad de dichas sacudidas.

Otra forma de implementarlo podría ser mediante el uso de la ubicación gps del dispositivo, similar al videojuego Pokemon GO, también mencionado al comienzo del proyecto, aunque esto evitaría la posibilidad de usar la aplicación en interiores, ya que se contabilizarían distancias a mayor escala que simples pasos. También añadiría más requisitos y permisos al dispositivo, por lo que posiblemente se optaría por el uso del giroscopio.

Con la implementación del contador de pasos, ya fuera mediante el giroscopio o el gps, podrían implementarse secciones en el juego; es decir, niveles sobre los cuales avanzar, de manera que cada cierta distancia apareciera un enemigo más fuerte, o jefe de zona.

El dispositivo original tenía otras muchas mecánicas, pero una que tenía gran importancia era el uso de energía, la cual era necesaria para utilizar ciertos personajes más poderosos durante los combates y que se obtenía entre otras formas, avanzando distancia en el juego. Esta sería la última de las implementaciones pensadas para el proyecto en caso de que se continuara con él, ya que este proyecto es una adaptación del dispositivo original que se ha querido realizar por la nostalgia hacia éste y sin ninguna finalidad lucrativa. Si se quisiera elaborar más, probablemente sería por una finalidad lucrativa y se optaría por un rediseño artístico del proyecto para evitar posibles problemas legales, implicando también muy posiblemente ligeros cambios en las funcionalidades.

Bibliografía

[1] EPDATA. ¿Cómo evoluciona la industria del videojuego en España? Fecha de última consulta 7 de Junio de 2022. https://www.epdata.es/datos/videojuegos-espanoles-espana-datos-graficos/292

[2] TECHNOCIO. Mobile gaming, la tendencia que llegó para quedarse. Fecha de última consulta 7 de Junio de 2022. https://technocio.com/mobile-gaming-la-tendencia-que-llego-para-quedarse/

[3] GOOGLE PLAY APPS. Motion Sensor. Fecha de última consulta 7 de Junio de 2022. https://play.google.com/store/apps/details?id=com.mtat.motiondetector&hl=es&gl=US

[4] GOOGLE PLAY APPS. QR Code Reader. Fecha de última consulta 7 de Junio de 2022. https://play.google.com/store/apps/details?id=tw.mobileapp.qrcode.banner&hl=es&gl=US

[5] GOOGLE PLAY APPS. Monitor de Frecuencia Cardíaca. Fecha de última consulta 7 de Junio de 2022. https://play.google.com/store/apps/details?id=com.repsi.heartrate&hl=es&gl=US

[6] GOOGLE PLAY APPS. Pokemon GO. Fecha de última consulta 7 de Junio de 2022. https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo&hl=es&gl=US

[7] WIKIPEDIA. Videojuego de estrategia por turnos. Fecha de última consulta 7 de Junio de 2022. https://es.wikipedia.org/wiki/Videojuego_de_estrategia_por_turnos

[8] WIKIPEDIA. Fire Emblem: Radiant Dawn. Fecha de última consulta 7 de Junio de 2022. https://es.wikipedia.org/wiki/Fire_Emblem:_Radiant_Dawn

[9] GOOGLE PLAY APPS. Fire Emblem Heroes. Fecha de última consulta 7 de Junio de 2022. https://play.google.com/store/apps/details?id=com.nintendo.zaba&hl=es&gl=US

[10] DIGIMONWIKI. Digivice. Fecha de última consulta 7 de Junio de 2022. https://digimon.fandom.com/es/wiki/Digivice

[11] UNIVERSIDAD DE SALAMANCA. Proceso Unificado. Fecha de última consulta 7 de Junio de 2022. https://repositorio.grial.eu/bitstream/grial/1948/1/7.%20PU-2020.pdf

[12] VEGAS HERNÁNDEZ, JESÚS MARÍA. Departamento de Informática de la UVa. Apuntes de la asignatura Planificación y Gestión de Proyectos. Curso 2019-2020.

[13] SÁNCHEZ MAYORAL, PABLO FEDERICO. Departamento de Organización de Empresas y Comercialización e Investigación de Mercados de la UVa. Apuntes de la asignatura Fundamentos de Organización de Empresas. Curso 2015-2016.

[14] TARIFALUZHORA. ¿Cuánto cuesta el kilovatio hora de luz (kWh) en España? Fecha de última consulta 7 de Junio de 2022. https://tarifaluzhora.es/info/precio-kwh

[15] GANAENERGÍA. ¿Qué Consumo nos Supone Utilizar el Ordenador? Fecha de última consulta 7 de Junio de 2022. https://ganaenergia.com/blog/que-consumo-nos-supone-utilizar-el-ordenador/

[16] TECNOXPLORA. ¿Sabes cuánto dinero cuesta al año cargar tu móvil? Fecha de última consulta 7 de Junio de 2022. https://www.lasexta.com/tecnologia-tecnoxplora/moviles/sabes-cuanto-dinero-cuesta-ano-cargar-movil_202008245f4411339a3b790001b25a91.html

[17] KEEPCODING. ¿Cuánto gana un Desarrollador en España? Fecha de última consulta 7 de Junio de 2022. https://keepcoding.io/blog/cuanto-gana-un-desarrollador-en-espana/

[18] LAGUNA, MIGUEL ÁNGEL. Departamento de Informática de la UVa. Apuntes de la asignatura Modelado de software. Curso 2017 -2018.

[19] WIKIPEDIA. Entity Component System. Fecha de última consulta 24 de Junio de 2022. https://en.wikipedia.org/wiki/Entity_component_system

[20] CRESPO GONZÁLEZ CARVAJAL, YANIA. Departamento de Informática de la Uva. Apuntes de la asignatura Diseño de software. Curso 2017 -2018.

[21] CASAS ALBERTOS, JOAQUÍN. Desarrollo de un videojuego en 2D para Android con Unity 3D. Trabajo de fin de grado. Grado en Ingeniería Informática de Servicios y Aplicaciones. 2015.

[22] DIGIMONWIKI. Gallery:Agumon. Fecha de última consulta 1 de Julio de 2022. https://digimon.fandom.com/wiki/Gallery:Agumon#Sprites_and_models

[23] DIGIMONWIKI. Gallery:Greymon. Fecha de última consulta 1 de Julio de 2022. https://digimon.fandom.com/wiki/Gallery:Greymon#Models_and_sprites

[24] DIGIMONWIKI. Gallery:Omnimon. Fecha de última consulta 1 de Julio de 2022. https://digimon.fandom.com/wiki/Gallery:Omnimon#Models_and_Sprites

[25] KAISADILLA. D-Tector-v2. Assets. Fecha de última consulta 1 de Julio de 2022. https://github.com/kaisadilla/D-Tector-v2/tree/master/Assets/Sprites

[26] PIXELSAGAS. Pixel Digivolve. Fecha de última consulta 1 de Julio de 2022. https://www.pixelsagas.com/?download=pixel-digivolve

[27] WIKIPEDIA. GitHub. Fecha de última consulta 2 de Julio de 2022. https://es.wikipedia.org/wiki/GitHub

[28] WIKIPEDIA. Astah*. Fecha de última consulta 2 de Julio de 2022. https://en.wikipedia.org/wiki/Astah*

[29] WIKIPEDIA. Visual Studio. Fecha de última consulta 2 de Julio de 2022. https://es.wikipedia.org/wiki/Microsoft_Visual_Studio

[30] WIKIPEDIA. Unity (motor de videojuego). Fecha de última consulta 4 de Julio de 2022. https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

[31] UNITY MANUAL. Order of execution for event functions. Fecha de última consulta 5 de Julio de 2022. https://docs.unity3d.com/2019.4/Documentation/Manual/ExecutionOrder.html

[32] S. SHAH, V. ADHIKARI, V. POKHRIYAL. Motion detection algorithm based on Background Subtraction. International Journal of Scientific & Engineering Research. Vol. 4. Issue 8. Aug.2013. Fecha de última consulta 6 de Julio de 2022. https://www.ijser.org/paper/Motion-detection-algorithm-based-on-Background-Subtraction.html

Anexos

Instalación de la aplicación

La instalación de la aplicación se puede realizar de manera directa desde el editor de Unity con la opción “Build and run” con la que instala el archivo al dispositivo conectado al PC que cumpla con las características asignadas al proyecto. En caso de querer instalarse en un dispositivo concreto, es posible configurarlo desde las opciones de la opción “Build settings”.

A continuación se muestra un enlace donde puede descargarse tanto los datos necesarios para trabajar con el proyecto, como el archivo apk comprimidos: https://github.com/DieVillUVa/D-Tector-TFG

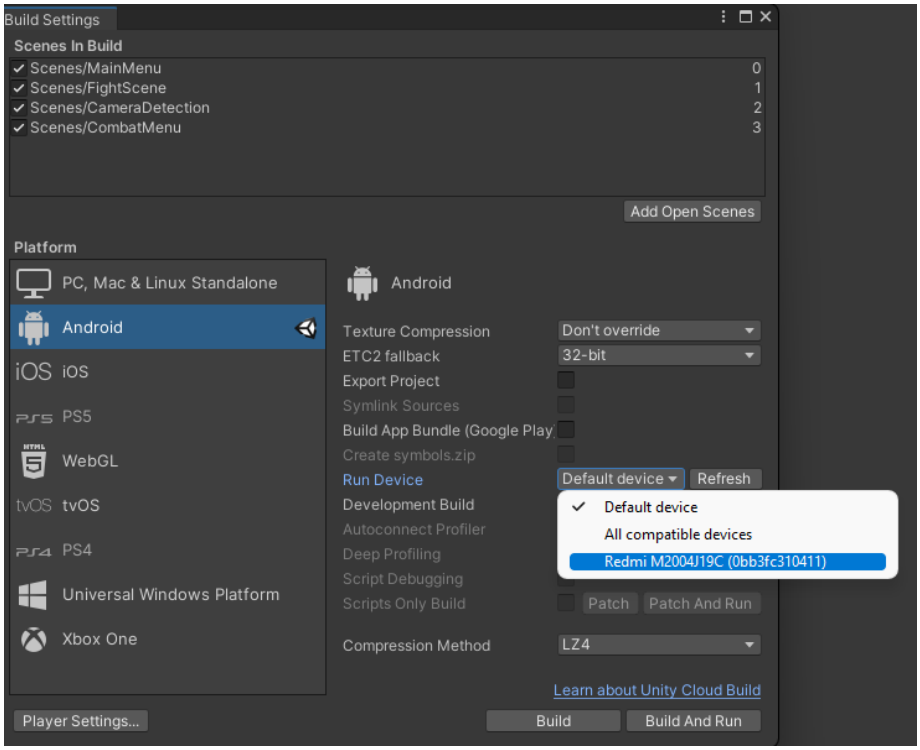

Ilustración 31: Configuración del dispositivo sobre el que construir la app

Esta es la forma más cómoda para instalar y testear la aplicación ya que simplemente necesita conectar el dispositivo al PC desde el menú del proyecto Unity, pero para ello es necesario tener acceso al proyecto.

Para instalar la aplicación desde un dispositivo externo, se crea un archivo apk durante la fase de building de la aplicación, que puede ser descargado e instalado.

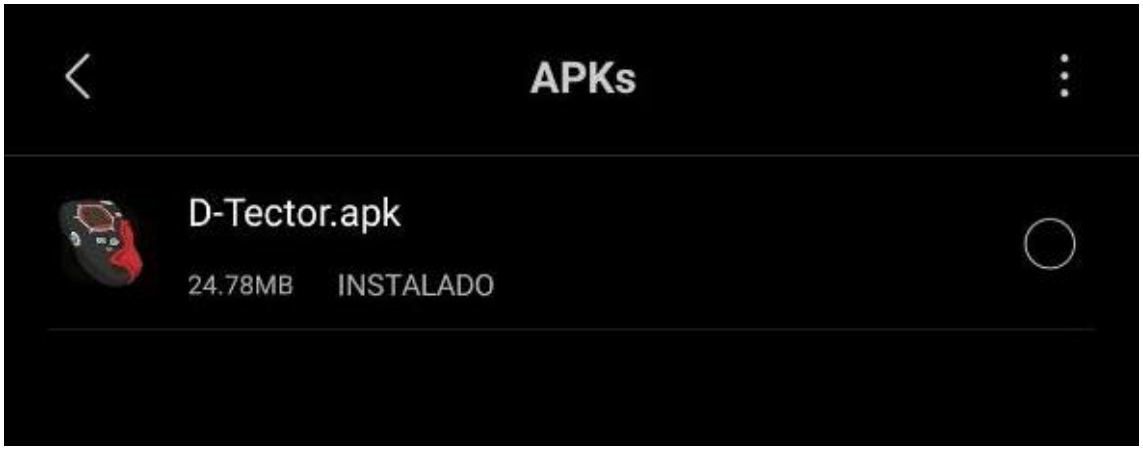

Ilustración 32: Archivo apk descargado e instalado en un dispositivo Android

Nota: Es posible que al tratarse de una aplicación desconocida sin ningún certificado de seguridad verificado, el sistema muestre alguna advertencia de peligro a la hora de descargar o instalar el archivo.

Una vez instalado el apk, la aplicación aparecerá en el dispositivo junto con el resto de apps y funcionará de manera normal sin la necesidad de acceder al proyecto ni al editor de Unity.

Manual de usuario

La aplicación consta de un menú principal sobre el que se muestran tres botones de dificultad, pulsando uno de ellos se iniciará el combate.

Dificultades y estadísticas

Cada dificultad tiene asignado un enemigo con unas estadísticas preestablecidas:

- **Fácil:**
  - Vida inicial: 105
  - Fuerza: 45
  - Velocidad: 40
  - Técnica: 50
  - Probabilidad de ataque Energy: 10%
  - Probabilidad de ataque Crush: 50%
  - Probabilidad de ataque Ability: 40%

- **Medio:**
  - Vida inicial: 160
  - Fuerza: 60
  - Velocidad: 40
  - Técnica: 70
  - Probabilidad de ataque Energy: 30%
  - Probabilidad de ataque Crush: 15%
  - Probabilidad de ataque Ability: 55%
- **Difícil:**
  - Vida inicial: 250
  - Fuerza: 60
  - Velocidad: 55
  - Técnica: 80
  - Probabilidad de ataque Energy: 45%
  - Probabilidad de ataque Crush: 25%
  - Probabilidad de ataque Ability: 30%

El personaje del jugador es una copia exacta del personaje de dificultad media. La idea es que el usuario pueda superar sin problemas la dificultad fácil sin ningún tipo de conocimiento del sistema. Ni valores de daño, ni el sistema de curación.

La dificultad media está pensada para que pueda superarse sin mucho conocimiento del juego, aunque es posible perder si se desconocen algunas mecánicas de éste, como la curación.

Para superar el modo difícil es necesario conocer los distintos valores de daño del sistema y el funcionamiento del sistema de curación. Está pensado para no poder ganarse de manera constante, ya que el enemigo tiene gran cantidad de vida y daño, pero el conocimiento de estos valores hacen que el nivel sea asequible para superarse en unos pocos intentos.

Menú de combate

Una vez seleccionado el modo de dificultad se mostrarán a los personajes que combaten y tras esto se nos mostrará el menú de combate.

En el menú de combate existen tres posibles acciones, accesibles mediante un menú de navegación representado por flechas ubicado en la parte inferior de la pantalla:

- **Atacar (Attack):** Esta acción iniciará la cámara para captar el tipo de ataque que quieres realizar y tras esto se iniciará el turno de combate entre el aliado y el enemigo.

- **Curar (Heal):** Al utilizar esta opción, el personaje aliado recuperará 90 puntos de vida, pero nunca superará el máximo de puntos inicial. Esta opción solo puede utilizarse una vez por combate y al utilizarla el enemigo no atacará ese turno, por lo que es conveniente intentar mantenerla hasta que el personaje aliado haya perdido suficiente vida como para recuperar el máximo de puntos que la acción ofrece.

- **Abandonar (Escape):** Al pulsar la opción el combate se abandona, volviéndose al menú principal. Esta opción puede ser utilizada en caso de que el usuario cambie de opinión y decida probar con otra dificultad diferente, o para reiniciar el combate sin tener que esperar a que este termine, si ha habido un mal comienzo.

Sistema de combate y selección de ataque

El juego tiene tres tipos de ataque; cada uno vence sobre otro independientemente de su poder, pero en caso de que ambos personajes utilicen el mismo tipo de ataque, el resultado se resolverá teniendo en cuenta el ataque con mayor poder:

- **Energy (Fuerza):** Este ataque utiliza la estadística de fuerza del personaje para calcular el daño inflingido. Este ataque vence sobre el tipo Ability, pero pierde contra Crush.
- **Crush (Velocidad):** Utiliza la velocidad asignada al personaje para realizar daño al enemigo. Vence sobre Energy y pierde sobre Ability.
- **Ability (Técnica):** El ataque utiliza el valor de la estadística técnica del personaje para calcular el daño sobre el rival. El ataque vence a Crush, pero pierde contra Energy.

Para elegir el ataque que se desea realizar por parte del aliado es necesario hacer una secuencia específica de pasadas sobre la cámara durante la fase de selección de ataque. Esta fase se divide en 3 secuencias de tiempo, cuyo final se representa visualmente con un cambio en la imagen mostrada, y sobre las que el usuario tiene la elección de pasar la mano sobre la cámara o no. En caso de que la cámara detecte una obstrucción de la luz sobre el foco, se asignará el valor 1, mientras que si el tiempo asignado para la secuencia termina sin detectar ninguna obstrucción, se asignará el valor 0. El ataque realizado dependerá del patrón obtenido:

- 1, 1, 1 (pasada, pasada, pasada): El personaje aliado realizará el ataque de tipo Energy.
- 1, 0, 1 (pasada, no pasada, pasada): El personaje aliado realizará el ataque Crush.
- 1, 1, 0 (pasada, pasada, no pasada): El personaje realizará el ataque tipo Ability.

Cualquier otro patrón de movimiento generará un ataque aleatorio.