Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN DE INGENIERÍA DE SOFTWARE

DIABERSE II

Alumna:
María José García Matías
Tutores:
César Pablo Gutiérrez Martínez (UVa)
Luis Vidal de la Rosa (Observatorio HP)

A mi familia

Agradecimientos

En primer lugar, quiero expresar mi más profundo agradecimiento a mi familia. Gracias por estar siempre a mi lado, proporcionándome su amor y comprensión en cada momento. Sin vuestra confianza y respaldo, no habría sido posible llegar hasta aquí.

También quiero agradecer a mis amigos, quienes han sido un pilar fundamental durante este proceso. Gracias por vuestra compañía, por los momentos de distracción necesarios y por animarme a seguir adelante cuando las cosas se ponían difíciles.

A mi pareja Edu, gracias por tu paciencia y comprensión. Tu apoyo emocional y tus palabras de ánimo han sido esenciales para superar los desafíos que se presentaron en el camino.

No puedo olvidar a mis compañeros de carrera, quienes se han convertido en grandes amigos. Gracias por los momentos compartidos, por las risas, por el apoyo mutuo y por hacer de esta experiencia universitaria algo inolvidable. Hemos aprendido y crecido juntos, y sin vosotros, este recorrido no habría sido el mismo.

Finalmente, quiero expresar mi gratitud a mis tutores de TFG. Su guía y orientación han sido cruciales para el desarrollo de este proyecto.

Gracias a todos.

Resumen

En los últimos años ha aumentando considerablemente el número de diagnósticos de diabetes entre los jóvenes y, aunque esta enfermedad permite llevar un ritmo de vida normal, supondrá un cambio significativo en su modo de vida.

El siguiente trabajo, realizado en colaboración con el observatorio HP-SCDS, ha consistido en el desarrollo de un videojuego 3D orientado al metaverso y en realidad virtual cuyo objetivo se centra en que estas personas que debutan en diabetes puedan aprender y comprender de una forma sencilla y amena cómo afecta esta enfermedad en su día a día. Así mismo, se contribuye a la mejora de la educación diabetológica, en la que aún queda mucho por avanzar.

El videojuego ofrecerá una experiencia inmersiva donde el usuario podrá elegir entre cinco niveles en los cuales deberá tomar diferentes decisiones para mantener su nivel de glucosa en un rango adecuado. De esta forma se pretende que los usuarios aprendan los conceptos básicos del control de la diabetes.

Palabras clave: diabetes, realidad virtual, metaverso, videojuego, debut diabético.

Abstract

In recent years, the number of diabetes diagnoses among young people has increased considerably. Although this disease allows them to lead a normal life, it will mean a significant change in their lifestyle.

The following work, carried out in collaboration with the HP-SCDS Observatory, has involved developing a 3D metaverse oriented video game in virtual reality whose objective is to enable these people who are new to diabetes to learn and understand in a simple and pleasan way how this disease affects their daily lives. It also contributes to the improvement of diabetes education, where there is still a long way to go.

The videogame will offer an immersive experience allowing users to choose five levels in which they must make various decisions to keep their glucose levels in an adequate range. In this way, it is intended that users learn the basic concepts of diabetes control.

Keywords: diabetes, virtual reality, metaverse, videogame, diabetic debut.

Índice general

Lista de figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

En la actualidad ha aumentado considerablemente el número de jóvenes que debutan en diabetes, esto les supone un gran cambio en su estilo de vida y deben aprender a vivir con esta enfermedad. Este TFG pretende, mediante un juego en realidad virtual, simular el comportamiento de su nivel de glucosa en el día a día facilitando su comprensión.

La **diabetes sacarina** [66], también conocida como *diabetes mellitus*, es una enfermedad crónica que se presenta cuando el páncreas no secreta suficiente insulina o cuando el cuerpo no utiliza adecuadamente la insulina[1] que produce. Existen diferentes tipos de diabetes [66]:

Tipo 1 o insulinodependiente Es la menos habitual y es más común en niños, adolescentes y jóvenes. Se caracteriza por una producción deficiente o inexistente de insulina debido a la destrucción autoinmune de las células *beta* del páncreas. Este trabajo estará dirigido a este tipo de diabetes.

Tipo 2 o insulinoresistente La sufren más del 95 % de personas con diabetes. Afecta a la forma en que el cuerpo utiliza el azúcar para obtener la energía, impidiendo que use la insulina de forma adecuada, lo que puede provocar un aumento de la glucosa en sangre si no es tratado.

Un nivel aceptable de glucosa en sangre es el que está dentro del rango 70-120, donde cualquier nivel por debajo de 70 se considera **hipoglucemia** y cualquier valor por encima de 120 se consideraría una **hiperglucemia**. Este rango depende del país aunque en España, por norma general, el rango es el mencionado anteriormente. Tanto estar por encima como por debajo del rango implica riesgos para la salud:

[1] Hormona que regula la glucemia, que es la concentración de glucosa en la sangre.

- **Hipoglucemia**: En los casos más graves provoca una pérdida del conocimiento impidiendo la ingesta de azúcares de absorción rápida. Al no poder ingerir este tipo de hidratos de carbono podría llevar a un coma diabético [48]. Los daños suelen ser producidos a corto plazo.
- **Hiperglucemia**: La glucosa en sangre cuando se dispara puede generar cetona, un compuesto orgánico tóxico. Si es persistente puede ocasionar complicaciones que afectan a los ojos, los riñones, los nervios y el corazón [49]. En este caso, los daños suelen ser a largo plazo.

Tanto la alimentación como la actividad física afectan al nivel de glucosa en sangre. Los **hidratos de carbono** [22], también llamados glúcidos, son nutrientes que proporcionan energía a nuestro organismo y están formados por oxígeno, hidrógeno y carbono. Se pueden clasificar de varias formas:

Según su estructura química [22]

- **Hidratos de carbono simples:** Están formados por pocas moléculas de azúcar. Engloba a los monosacáridos, como la glucosa o la fructosa, y a los disacáridos, como la lactosa o la sacarosa. Son de sabor dulce y de rápida absorción intestinal.
- **Hidratos de carbono complejos:** Son los polisacáridos como el almidón y el glucógeno. No suelen ser dulces y al contrario que los simples tienen una absorción más lenta.

Según la velocidad de absorción [26]

- **De absorción rápida:** Apenas necesitan procesos a nivel interno para ser absorbidos. En este caso la glucosa pasa muy rápido a la sangre y esto genera picos de glucemia[2] elevados. En este grupo se encuentra el azúcar refinado, los zumos, la miel, los dulces, etc.
- **De absorción lenta:** A diferencia de los anteriores, requieren de mayor tiempo para ser absorbidos debido a que el organismo debe transformarlos a hidratos simples. Estos no producen picos de glucemia elevados. Algunos ejemplos de fuentes de alimentos que contienen este tipo de hidratos son los cereales, las harinas, las pastas, las legumbres, los arroces, etc.
- **De absorción moderada**: Contienen tanto hidratos de carbono simples como complejos. Algunas fuentes de alimentos en las que se encuentran este tipo de hidratos son las frutas y el pan blanco.

Por lo tanto, las personas que sufren diabetes tienen que aprender a contar las raciones[3] de hidratos de carbono que ingieren para poder mantener un índice glucémico saludable y estable.

[2] Los picos de glucemia son elevaciones descontroladas del nivel de glucosa en un periodo corto de tiempo.
[3] Una ración equivale a 10 gramos de hidratos de carbono

Por otro lado, deben aprender a calcular el número de unidades de insulina que necesitarán para metabolizar esa ingesta y continuar manteniendo el azúcar en sangre estable, evitando posibles hipoglucemias o hiperglucemias. En diabetes de tipo 1 deberá aplicarse inyecciones de insulina diariamente y, a continuación se explican los diferentes tipos y sus características: [96]

- **Insulina de acción prolongada**, (insulina Glargina, insulina Detemir) se absorbe lentamente y se usa para controlar el azúcar en sangre durante la noche en periodos de sueño, mientras se está en ayunas y entre comidas. Tiene una duración de 24 horas en el cuerpo.
- **Insulina de acción intermedia**, (insulina humana NPH, insulina premezclada) se utiliza igual que la insulina de acción prolongada pero se necesitan más dosis al día ya que dura menos. Empieza a hacer efecto entre 1 a 2 horas después de su aplicación y el pico se obtiene a partir de la cuarta a sexta hora.
- **Insulina de acción rápida**, (insulina Aspart, insulina Lyspro, insulina Glulisina) se absorbe rápidamente y se utiliza para regular el azúcar en sangre que se produce en las digestiones de las comidas principales del día y para corregir cuando haya picos de azúcar descontrolados. Comienza su efecto a los 5-15 minutos, el pico se da entre 1 a 2 horas y tiene una duración de unas 4 a 6 horas.

En la Figura 1.1 podemos ver los diferentes rangos de acción de los tipos de insulina que se han explicado anteriormente. La "Normal" hace referencia a la insulina que produce una persona sin diabetes.

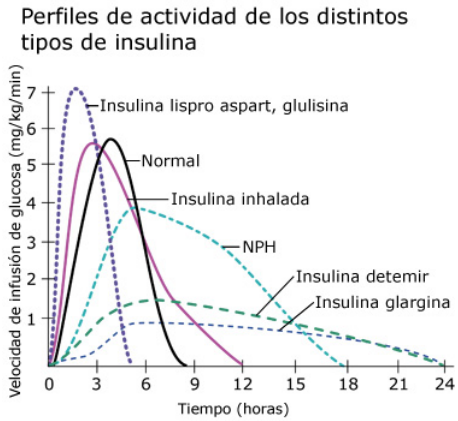

Figura 1.1: Comparación de los diferentes efectos de cada tipo de insulina.

1.1.1. Introducción a la realidad virtual

La realidad virtual, del inglés *Virtual Reality* (VR), ha cogido mucha importancia en nuestro día a día y, en la actualidad, ya se puede utilizar en cualquier lugar con un hardware

básico. Es una tecnología que para algunos puede parecer futurista pero que desde los años ochenta ha ido creciendo y mejorando hasta como la conocemos hoy en día.

¿Qué es?

La realidad virtual [39] es la creación de un entorno o escenario ficticio con apariencia realista y que consigue trasladarnos a cualquier lugar o situación (inmersión). Se necesitarán unas gafas o unos cascos de realidad virtual para poder simular de forma muy realista que nosotros somos los protagonistas del juego (simulación 3D). A parte se permite el uso de otros dispositivos para poder movernos e interactuar con el entorno que se está simulando como mandos o *joysticks* (interactividad).

Realidad virtual vs realidad aumentada

La diferencia más significativa entre la realidad virtual y la realidad aumentada (AR, del inglés *Augmented Reality*) está en que esta última utiliza un escenario real incluyendo objetos no reales, y la realidad virtual tiene un 100 % del escenario ficticio. Un ejemplo de realidad aumentada podría ser el *Pokemon Go* como se muestra en la Figura 1.2, donde se ve como se mezcla el escenario real con el *pokemon* ficticio.

Figura 1.2: Ejemplo de Realidad aumentada en el *Pokemon Go* [68]

Además, existe la **realidad mixta** que es una mezcla de los mejores aspectos de la realidad virtual y de la realidad aumentada, permitiendo interactuar con objetos reales dentro de un mundo totalmente digital.

Dispositivos de realidad virtual

Las gafas de realidad virtual [41], también denominadas cascos de realidad virtual, son dispositivos físicos que se colocan en la cabeza y que tienen la capacidad de reproducir vídeos o imágenes haciendo que el usuario perciba que forma parte de lo que se esta reproduciendo. Esto se consigue debido a que el dispositivo está muy cerca de los ojos, obteniendo así un campo de visión muy amplio.

Este tipo de dispositivos cuenta con dos pantallas, una para cada ojo, donde se reproducen imágenes muy similares. A esto se le conoce como **visión estereoscópica** y, aunque son dos pantallas, consigue que el cerebro lo perciba como una única imagen tridimensional. Es muy importante que las imágenes que se reproduzcan sean en tres dimensiones para que el usuario tenga la sensación de profundidad.

Además, estos dispositivos para proporcionar una experiencia aún más realista, tienen incorporados una serie de sensores. Gracias a estos **sensores**, las imágenes que se reproduzcan se moverán siguiendo los movimientos del usuario. De esta forma el usuario sentirá la experiencia de una forma aún más realista. El sensor que más se suele utilizar para detectar los movimientos realizados con la cabeza son los giroscopios. Otro tipo de sensor muy utilizado es el acelerómetro.

A continuación se describen los diferentes tipos de gafas de realidad virtual existentes [70]:

Gafas de realidad virtual para móviles: Son las más económicas gracias a que están preparadas para su utilización junto con un dispositivo móvil. No contienen ningún hardware y aunque no incorporan ninguna pantalla (se utiliza la del dispositivo móvil) si que están equipadas con unas lentes que permiten dar un efecto 3D a dicha pantalla. Este tipo es el único que no contiene sensores ni mandos.

Gafas de VR para consola: Especialmente diseñados para las videoconsolas, contienen conexiones con un procesador externo que será el de la consola en cuestión a la que estarán conectadas por cable. Esto significa que utilizan el hardware de la videoconsola para poder generar el contenido. Las gafas *PlayStation VR* diseñadas para la *PlayStation* serían de este tipo.

Gafas de VR para ordenador: Diseñado para su uso junto con un ordenador, son las más potentes gracias a que permiten utilizar el hardware de los equipos y utilizando configuraciones avanzadas se puede mejorar mucho la experiencia de usuario. En este caso también se conectarán por cable y un ejemplo de este tipo serían las *Oculus Rift*.

Gafas de VR autónomas: A diferencia de las anteriores no están conectadas por cable a ningún otro dispositivo ya que el hardware que poseen es más que suficiente para su correcto funcionamiento. En este tipo encontramos las *Oculus Go* o las *Meta Quest 2*.

Junto con las gafas de realidad virtual se suelen utilizar una serie de accesorios para la interacción con el entorno virtual mejorando así la experiencia del usuario. Los accesorios más utilizados son los **controladores de movimiento** que permiten al usuario interactuar con los objetos virtuales del entorno de una manera precisa y fácil. También existe otra variante de estos, denominados controlares de movimiento omnidireccionales, que además permiten a los usuarios moverse físicamente.

Otro tipo de accesorios utilizados para la realidad virtual son los ratones 3D, los rastreadores ópticos, los guantes por cable o los dispositivos olfativos.

1.1.2. Metaverso

El metaverso [46] es un espacio virtual compuesto por entornos virtuales 3D donde se pueden combinar diferentes tecnologías de realidad virtual, realidad aumentada y otras tecnologías inmersivas utilizando Internet como red subyacente. También se puede describir como un universo digital paralelo que trata de extender el mundo físico mediante el uso de dichas tecnologías.

En este espacio virtual, los usuarios pueden aprender, socializar y jugar con sus avatares digitales gracias a que pueden interactuar con objetos del entorno, y entre ellos. Algunas características importantes son:

- **Persistencia:** Este entorno nunca se "apaga" ni se "reinicia", continúa ininterrumpidamente. Esto también implica que los datos no son borrados.
- **Síncrono:** Es una experiencia que es consistente para todos los usuarios y en tiempo real.
- **No hay límite de usuarios:** El metaverso permite a cualquier persona formar parte de él, dando a cada usuario esa sensación de presencia al participar en diferentes eventos o actividades al mismo tiempo que otros usuarios.
- **Economía:** Dentro de este universo digital existe trabajo que produce un valor reconocido por otros. Se puede comprar o vender activos digitales que pertenecen exclusivamente al avatar y que pueden ser comercializados en el metaverso.
- **Integración del mundo digital con el físico:** Esta es una experiencia que intenta unificar y envolver ambos mundos.
- **Interoperabilidad:** Ofrece una interoperabilidad de datos nunca antes vista en el mundo digital, donde ahora un diseño de un arma en un juego podrá ser utilizado en cualquier otro juego o compartido con otros usuarios.
- **Creado por una gran variedad de contribuyentes:** El metaverso tiene una gran variedad de contenidos y experiencias que han sido creadas tanto por personas individuales como empresas con un enfoque más comercial manteniéndolo en una continua evolución.

Para poder acceder a este espacio virtual [43] no es necesario un dispositivo en específico. Existen diferentes pasarelas para acceder a cada plataforma desde diferentes dispositivos. Algunas opciones para acceder al metaverso son unos **cascos de realidad virtual o aumentada**, desde **consolas de videojuegos** (aunque estás podrían tener ciertas limitaciones si no se cuenta con gafas de realidad virtual), **ordenadores** con gráficos y con al menos 2 Gb de RAM y también existen **ciertas aplicaciones móviles** que permiten el acceso al metaverso sin hardware adicional.

1.2. Motivación

Esta idea de trabajo de fin de grado surgió propuesta por la empresa HP-SCDS como una extensión a Diaberse, entorno de realidad virtual en el cual un avatar a través de comandos de voz enseñaba cómo cuidar y controlar la diabetes a través de vídeos educativos. Esta extensión se crea con la finalidad de que, principalmente, niños/as y adolescentes que están debutando en diabetes puedan aprender a controlar su nivel de glucosa en la vida diaria a través de un videojuego.

Entre 1990 y 2019 ha aumentado la diabetes más de un 55 % entre la población adolescente y joven [74]. Además, entre 2019 y 2020, primer año de la pandemia, hubo un incremento del 14 % en diabetes de tipo 1 en niños y adolescentes menores de 19 años y la incidencia aumentó aún más en el segundo año de la pandemia, un 27 % más que en 2019 [40].

Para los afectados el diagnóstico de esta enfermedad supone un cambio importante en su estilo de vida ya que a partir de ese momento deben realizar un seguimiento constante en la alimentación, actividad física que practican diariamente, pero sobre todo, en su índice de glucosa en sangre. En cuanto a este último punto, para mantenerla en un rango de valores aceptable deben adquirirse conocimientos que no son sencillos y es muy importante, sobre todo al comienzo de la diabetes, que tengan acceso a recursos que les ayuden a entender como afecta la insulina en el nivel de glucosa y así conseguir un mayor control de esta enfermedad crónica.

Por todo ello es importante visibilizar la necesidad de crear herramientas que ayuden a las personas diagnosticas a adaptarse a su nueva vida.

1.3. Grupos objetivo

Este proyecto ocupa el segmento de mercado de personas que han sido diagnosticadas recientemente en diabetes y quieren aprender sobre esta enfermedad de una forma divertida e inmersiva. Por lo tanto, los beneficiarios principales de este videojuego son personas jóvenes hasta mediana edad, entre los 8 y 30 años, que acaban de debutar en diabetes y que tengan cierta familiaridad con dispositivos tecnológicos.

Además, otro grupo objetivo son los familiares y amigos de estas personas que desean poder comprender la enfermedad para ayudarles cuando lo necesiten.

1.4. Objetivos

El objetivo principal es el desarrollo de un videojuego 3D educativo que sirva como herramienta de aprendizaje a personas que han sido diagnosticadas diabéticas para comprender el funcionamiento de su nivel de glucosa en sangre.

1.4.1. Objetivos docentes

Los objetivos docentes de este proyecto se centran en tratar de aplicar los conocimientos adquiridos a lo largo de la carrera universitaria.

Se seguirá el marco de gestión de proyectos de metodología ágil, Scrum estudiada en la asignatura de Planificación y Gestión de Proyectos y se utilizará para desarrollar un proyecto software de inicio a fin, pasando por todas las fases de desarrollo de un proyecto de desarrollo software profesional, poniendo a prueba los conocimientos adquiridos en asignaturas como Fundamentos de Ingeniería de Software, Modelado de Software y Diseño de Software.

Para el control de versiones del proyecto se hará uso de Git, estudiado en la asignatura de Tecnologías para el Desarrollo de Software y para el diseño de la interfaz de usuario se aplicarán los conceptos estudiados en la asignatura Interacción Persona-Computadora.

Por último, se pretende elaborar y defender una presentación pública del trabajo realizado, poniendo en práctica lo aprendido en Profesión y Sociedad.

1.4.2. Objetivos del proyecto

El objetivo del proyecto es desarrollar un videojuego 3D orientado al metaverso y en realidad virtual para que la persona con diabetes pueda aprender y entender de una forma sencilla y divertida cómo tener un control de las raciones de hidratos de carbono que ingieren, las unidades y tipos de insulina que se inyectan y la actividad física que realizan. Así aprenderá cómo le afecta cada uno de estos factores a su barra de glucosa, que simulará el nivel de glucosa del usuario, el cual deberá estar entre 70 y 120. Dicha barra es un símil a las barras de vida que aparecen en otros videojuegos.

De esta forma se pretende ayudar a las personas a comprender de una forma visual, divertida y atractiva qué efectos tiene comer una ración de hidratos de carbono, qué efecto tiene aplicarte insulina o qué sucede cuando haces ejercicio.

El videojuego se desarrollará en una pista de atletismo y en su fase inicial se implementarán 5 niveles donde el protagonista tendrá que tomar diferentes decisiones como la ingesta de hidratos de carbono o la aplicación de insulina para mantener su barra de glucosa en unos niveles saludables. Al subir de nivel se irán añadiendo nuevos conceptos sobre la diabetes, aumentando así la complejidad de los mismos.

1.4.3. Objetivos personales

Además este trabajo de fin de grado pretende conseguir otros objetivos a nivel de formación personal:

1. Aprender el lenguaje de programación C# junto con el motor de videojuegos Unity.

2. Mejorar mi habilidad de redacción, enfocándome en expresar ideas de manera clara, concisa y efectiva.

3. Adquirir conocimientos sobre la diabetes para apoyar y ayudar a personas con esta enfermedad.

1.5. Estudio de alternativas

En la actualidad no existen alternativas que supongan claros competidores para el videojuego objetivo, pero si que existen algunas apps/videojuegos con un objetivo similar. A continuación, se explicarán de manera breve y se realizará una comparativa con SugarRace, nombre elegido para denominar a la segunda parte de Diaberse.

1.5.1. Diabetes Voyager

Este es el único juego que pretende promover el aprendizaje y comprensión del funcionamiento de la diabetes en realidad virtual.

El juego tiene una perspectiva que permite al usuario hacer un viaje por dentro del cuerpo de una persona con diabetes, dando la posibilidad de visitar el corazón, el cerebro y el sistema vascular. Además, contiene una serie de desafíos clave muy relacionados con el control de la diabetes donde el usuario puede interactuar mediante una serie de sensores con el entorno, aprendiendo sobre la diabetes de una manera bastante innovadora [8].

En la Figura 1.3 se puede ver una escena dentro del cuerpo de la persona con diabetes con la puntuación y otros datos en la pantalla.

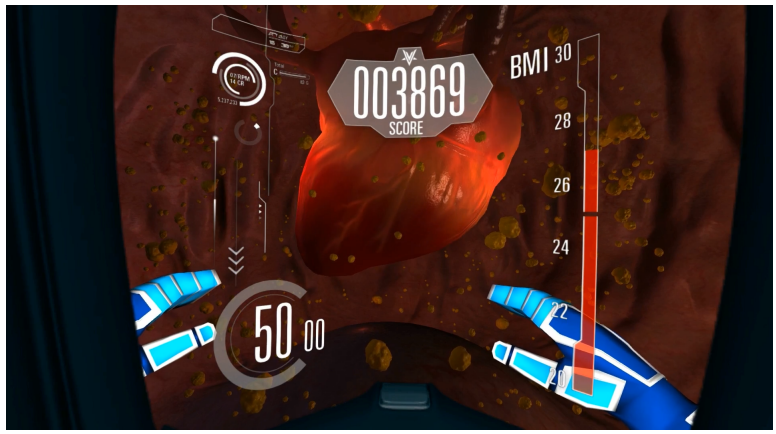

Figura 1.3: Escena de Diabetes Voyager [8]

1.5.2. Diabegame

Diabegame [3] es un juego que está pensando principalmente para jugar con el móvil aunque también es compatible con tablets y ordenadores. Esta dirigido a personas con diabetes tipo 1, mayores de 14 años y familias de menores con diabetes. Consta de 7 etapas, cada una con un tema diferente sobre la diabetes y el objetivo es actualizar tus conocimientos sobre diabetes de una forma divertida.

1.5.3. GlucoZor

GlucoZor [44] es una aplicación para dispositivos móviles y tablets enfocado a niños de entre 8 y 12 años, que tiene como objetivo cuidar de un dinosaurio diabético, darle comida y jugar con él mientras el niño está pendiente de su nivel de glucosa. Pretende ayudar a personas que sufren esta enfermedad a mejorar su cuidado personal y a ayudarlas para que puedan mantener su autonomía.

En la Figura 1.4 se aprecia al dinosaurio y en la parte superior derecha se encuentra la barra que marca su nivel de azúcar en sangre.

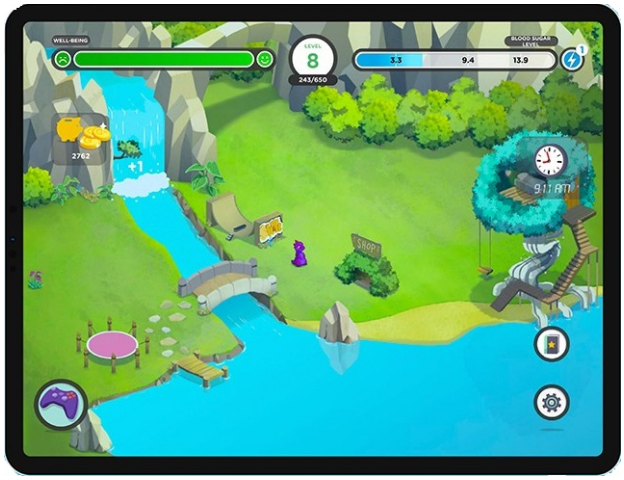

Figura 1.4: Escena de GlucoZor [44]

1.5.4. DART

DART (*Diabetes-Augmented Reality Training*) [20] es un proyecto europeo que fomenta la actividad física en personas con diabetes utilizando la realidad aumentada en una aplicación móvil. Es un proyecto pionero que utiliza la realidad aumentada para ofrecer una experiencia envolvente mientras se aprende sobre el funcionamiento de la diabetes y además lo aprovechan para mejorar la salud de estas personas.

Está dirigido a niños, adolescentes y adultos de diabetes tipo 1 y 2 pero también a sus familiares, profesores de educación física o entrenadores.

El juego tiene un entrenador personal en forma de holograma donde nos va guiando en la realización de ejercicios específicos según el tipo de diabetes y características de cada persona, y cuya intensidad, frecuencia y velocidad va aumentando a lo largo del juego. Estos ejercicios son explicados por voluntarios de diferentes edades mediante vídeos con el objetivo de motivar a personas con diabetes a llevar una vida más saludable.

1.5.5. Carb Counting with Lenny

Este juego es una aplicación móvil gratuita que ayuda a los niños a convertirse en expertos sobre el conteo de hidratos de carbono. Lenny es un león que guía tanto a niños como a sus padres a través de una serie de juegos educativos sobre la diabetes.

Carb Counting with Lenny ofrece explicaciones sencillas con un diseño simple y atractivo con cuatro modos de juego: comparar y adivinar los hidratos de carbono, preparar la comida y aprender que alimentos tienen hidratos de carbono como se puede ver en la Figura 1.5. No utiliza términos complicados por lo que es apropiado para niños [24].

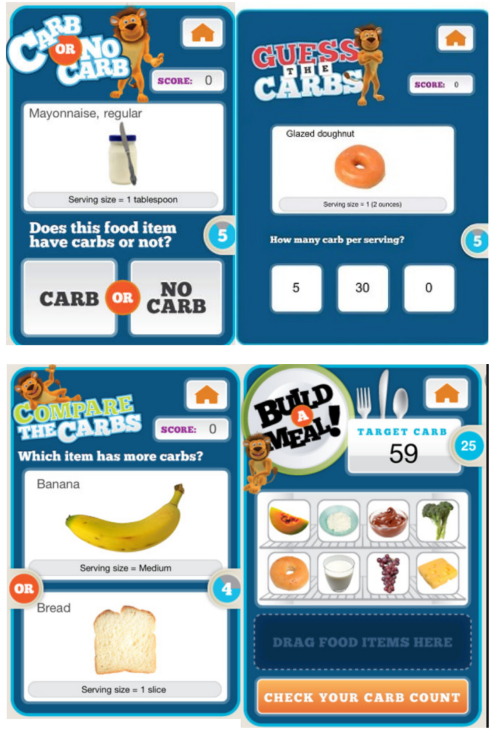

Figura 1.5: Diferentes modos de juego de Carb Counting with Lenny [11]

1.5.6. Comparativa con SugarRace

Una vez vistas todas las alternativas que existen hasta la fecha podemos compararlas con este proyecto. Aunque todas tienen como objetivo principal educar sobre la diabetes existen grandes diferencias entre ellas.

Las plataformas Lo óptimo es poder acceder al juego desde cualquier plataforma y así llegar al máximo número de jóvenes posible. SugarRace, a diferencia de algunas de las alternativas anteriores, será un juego en realidad virtual, esto podría parecer un problema pero en la actualidad es posible jugar a este tipos de juegos en el móvil junto con un mando Bluetooth, como el de la *Nintendo Switch* o el de la *Play Station 5* y unas gafas de realidad virtual, desde las más sencillas y baratas como *Google Cardboard* [38], hasta las más sofisticadas y caras como las *Samsung Gear VR* [71] o las *Oculus Go* [23].

Las dinámicas y mecánicas de juego [62] Las dinámicas de juego son aquellas necesidades e inquietudes humanas que motivan a las personas a continuar jugando. Para alcanzarlas se implementan diferentes mecánicas de juego, que son las normas que tiene todo juego y que tienen que cumplirse o llevarse a cabo para poder alcanzar el objetivo final.

Estas necesidades deben estar integradas de forma creativa y coherente para así poder captar la atención del usuario. Algunas dinámicas son las recompensas, los logros o la competición.

En SugarRace se implementará un sistema de obtención de puntos que se irán adquiriendo o perdiendo a lo largo de la carrera dependiendo del nivel de glucosa en sangre o de los objetos que se van recolectando. También habrá diferentes niveles donde la dificultad irá aumentando progresivamente.

Por otra parte, un sistema de obtención de logros al llegar a los puntos de control y al subir de nivel resultaría interesante para el jugador, pero esta funcionalidad solo se implementará si el tiempo de desarrollo avanza mejor de lo esperado.

SugarRace junto con Diabetes Voyager son los únicos juegos educativos en realidad virtual sobre la diabetes en la actualidad pero su forma de juego es totalmente diferente. Diabetes Voyager tiene una perspectiva que pretende educar mediante un viaje por dentro del cuerpo humano. SugarRace pretende dar al usuario una experiencia lo más realista posible en cuanto a realizar actividad física y a las diferentes decisiones que deberá tomar en el día a día sobre los hidratos de carbono que debe ingerir y la insulina que debe aplicarse de una forma mucho más dinámica que ayudará a la obtención de usuarios.

1.6. Estructura de la memoria

La estructura de la memoria será la siguiente:

- **Capítulo 2: Estado del arte.** Exposición de la situación actual de la diabetes y de la realidad virtual dentro del ámbito de la salud.

- **Capítulo 3: Tecnologías empleadas.** Descripción de las diferentes herramientas utilizadas para el desarrollo del proyecto.
- **Capítulo 4: Requisitos del cliente.** Especificación de los diferentes requisitos impuestos por el Observatorio HP-SCDS.
- **Capítulo 5: Planificación.** Presentación de la metodología de trabajo utilizada en el proyecto junto con la planificación y estimación inicial de los tiempos de desarrollo. También se expondrán los requisitos en forma de historias de usuario, los recursos, presupuestos y el plan de riesgos asociados al proyecto.
- **Capítulo 6: Análisis.** Adaptación de los requisitos del cliente a requisitos de producto, especificación de los actores, casos de uso y escenarios del proyecto.
- **Capítulo 7: Diseño.** Explicación de la arquitectura y patrones de diseño utilizados además de los diagramas que representan lo desarrollado.
- **Capítulo 8: Implementación del proyecto.** Explicación de diferentes conceptos generales de Unity además de explicaciones de cómo se han implementado las diferentes historias de usuario.
- **Capítulo 9: Pruebas.** Descripción de las diferentes pruebas realizadas para comprobar el correcto funcionamiento del videojuego desarrollado junto con pruebas de usabilidad y jugabilidad.
- **Capítulo 10: Seguimiento del proyecto.** Descripción de los diferentes *sprints* especificando las tareas desarrolladas, la planificación y retrospectiva de cada uno de ellos.
- **Capítulo 11: Conclusiones y líneas futuras.** Conclusiones obtenidas después de haber realizado el proyecto. También se expondrán algunas propuestas de mejora que se podrán realizar en un futuro.
- **Apéndice A: Manual de usuario.** Descripción de la interfaz de usuario de las diferentes escenas del videojuego.
- **Apéndice B: Manual de despliegue y ejecución.** Especificación de requisitos del sistema y pasos para el despliegue y ejecución del proyecto.

Capítulo 2

Estado del arte

La educación en salud y la tecnología en el tratamiento de la diabetes en niños y niñas son los temas centrales de este estudio. A continuación se presenta un resumen de la investigación previa sobre este grupo de población, los juegos de realidad virtual aplicados en este ámbito y las herramientas educativas para el manejo de la diabetes existentes.

2.1. Diabetes en la actualidad

En 2023, alrededor de 463 millones de personas entre 20 y 79 años padecían diabetes. Esto representa al 9,3 % de la población mundial de ese grupo de edad y se prevé que para 2030 aumente un 10,2 % llegando a 578 millones de personas [45]. Esta previsión significa que más de 100 millones de personas van a recibir este diagnóstico, lo que al principio les puede resultar abrumador y surjan muchas preguntas sobre cómo será su vida a partir de ese momento [10].

Hay que tener en cuenta que la diabetes es una afección crónica de larga duración y que por lo tanto, su cuidado conlleva un trabajo diario muy importante. Al ser diagnosticado de diabetes existirá un equipo de atención médica que ayudará a esta persona a aprender sobre el cuidado diario[1] y que le facilitará información sobre todas las herramientas de ayuda disponibles para esta enfermedad. Y aquí es donde surge el primer problema, dar esta información a los niños/as.

La mayoría de diagnósticos de diabetes de tipo 1 son en niños y niñas los cuales con una explicación de un equipo médico puede que no logren entender lo que realmente significan muchos de los conceptos relacionados con la diabetes. Este trabajo de fin de grado pretende complementar la explicación de los médicos con lo que más les gusta, jugar.

[1] Este proceso es conocido como "debut diabético" y en él, el niño o la niña permanecen ingresados durante unos cinco días en los cuales aprenden como controlar la diabetes mientras el doctor o doctora monitorizan el progreso.

La realidad virtual consigue acercar los conceptos más abstractos, de manera que los niños llegan a comprender mucho mejor todos los conceptos nuevos y además les será mucho más fácil recordarlos a largo plazo [61]. Es una forma de reforzar todas las explicaciones del equipo de atención médica de una forma lúdica captando por completo su atención y concentración acelerando su aprendizaje a través de la motivación intrínseca.

Además, se pretende desmitificar los siguientes mitos respecto a la diabetes [50]:

"Los diabéticos deben seguir una dieta especial" Las personas con diabetes consumen los mismos alimentos que cualquier otra persona. Se recomienda cuidar la alimentación al igual que se recomienda al resto de personas. Un plan de alimentación equilibrado junto con un estilo de vida saludable ayudará a controlar la diabetes.

"Los diabéticos no pueden comer azúcar" No está prohibido y pueden consumirlo siempre y cuando se planee. Pueden consumir cantidades pequeñas en lugar de otros hidratos de carbono o si toma insulina se tendrá que aplicar una dosis más alta de lo normal.

"Los diabéticos que usan insulina no manejan adecuadamente el nivel de glucosa" Las personas con diabetes tipo 1 deben utilizar insulina ya que su cuerpo no produce esa hormona y la diabetes de tipo 2 es progresiva por lo que irá produciendo cada vez menos. Entonces, la insulina es necesaria para mantener el azúcar en sangre en un rango saludable.

"Los diabéticos no pueden hacer deporte ni cualquier otra actividad física" De hecho hacer ejercicio de manera regular es una parte muy importante para el control de la diabetes. Ayuda a impulsar la sensibilidad del cuerpo a la insulina y se recomienda hacer por lo menos 150 minutos de ejercicio moderado, como caminar rápido, por semana.

SugarRace enseñará que no hay una dieta específica que se deba seguir, sino que en cada momento, dependiendo de varios factores como el nivel de glucosa, la insulina que se ha aplicado o de la realización de ejercicio, hay que contar correctamente las raciones de hidratos de carbono que debemos ingerir para mantener nuestro nivel de glucosa en sangre en un rango saludable. Además, se enseñarán los efectos que tiene la insulina y el deporte en el jugador mediante la barra de glucosa, que como ya se ha explicado representará el nivel de glucosa en sangre.

2.2. Realidad virtual para la enseñanza de la salud

La realidad virtual ha emergido como una herramienta prometedora en el campo de la educación en salud al proporcionar entornos inmersivos y experiencias interactivas que pueden mejorar la comprensión y el compromiso de los usuarios. El uso de juegos en VR se ha estudiado para una variedad de razones, como enseñar sobre enfermedades crónicas, promover estilos de vida saludables y para el manejo del dolor.

Por ejemplo, en el Hospital Universitario La Paz en Madrid [98] utilizan la realidad virtual con pacientes infantiles a los que se les va a realizar un trasplante de hígado. Existen 3 fases diferentes:

Explicación del procedimiento preoperatorio En esta primera fase el objetivo es disminuir la incertidumbre del paciente y de la familia enseñándoles mediante gafas de realidad virtual dónde van a estar el día del trasplante, cómo y por dónde van a bajar al quirófano, qué se hará allí... Al vivir esta experiencia, el día del trasplante ya cuentan con cierto grado de familiaridad con el procedimiento por lo que disminuye la ansiedad del paciente y de la familia.

Distracción En la segunda fase su uso se aplica en procesos que son dolorosos como quitar o poner drenajes. El paciente infantil al estar concentrado en la experiencia inmersiva en VR se consigue centrar su atención en otra tarea ayudando a reducir su sensación de dolor.

Formación y concienciación sobre el tratamiento En esta última fase, la realidad virtual se utiliza para enseñar, tanto al paciente como a los familiares, el tratamiento y cuidados que deben aplicar tras la operación así como nuevos hábitos y restricciones a su actividad normal.

En cuanto a la formación, ya no solo se centra en pacientes y familiares sino que también, a los futuros o actuales profesionales de la salud. Existen experiencias donde los estudiantes deben completar unas tareas desde la perspectiva de una persona de avanzada edad. De esta forma, se incrementa la comprensión de los problemas que tienen estas personas y aumenta la empatía de los estudiantes hacia las personas mayores.

También se utiliza para que los residentes de un hospital dispongan de un entorno de simulación realista pero seguro, con un alto nivel de inmersión que les permita fallar y aprender del error sin generar consecuencias.

Capítulo 3

Tecnologías empleadas

En este capítulo se llevará a cabo el desarrollo teórico de las tecnologías que han sido utilizadas para la realización de SugarRace. Para una mejor organización se han clasificado en tres secciones: tecnologías de desarrollo, tecnologías de comunicación y gestión del proyecto y tecnologías de documentación y diseño.

Las tecnologías que se van a explicar a continuación no son todas de libre elección, es decir, existen tecnologías de uso obligatorio que han sido establecidas por los requisitos de HP-SCDS como es el caso del uso de GitLab para el control de versiones del proyecto o Unity como motor de juegos.

3.1. Tecnologías de desarrollo

Las tecnologías de desarrollo (son marcas registradas) engloban aquellas tecnologías que se han utilizado para la programación del proyecto.

3.1.1. Unity

Unity a grandes rasgos es una plataforma para el desarrollo de videojuegos que con el tiempo se ha transformado en el motor de juegos multiplataforma más popular del mundo [78]. Nace en 2005 por Unity Technologies con el objetivo de que cualquier persona pudiera aprender, utilizar y contribuir en su motor de juegos.

Figura 3.1: Logo de Unity [105].

En este proyecto fue HP-SCDS quien estableció en los requisitos que se debía utilizar para el desarrollo del videojuego.

XR Plugin Management

XR Plugin Management [91] es una extensión de Unity donde se proporcionan complementos XR para administrar y ayudar al desarrollador con la carga, inicialización, configuración y soporte de compilación para poder utilizar complementos XR en nuestro proyecto. Ha sido utilizado para poder implementar la realidad virtual correctamente en el proyecto.

XR Interaction Toolkit

XR Interaction Toolkit [82] proporciona una gran variedad de herramientas para poder crear experiencias de realidad virtual y realidad aumentada. Está basado en componentes y crea un marco donde las interacciones 3D y UI estarán disponibles desde los eventos de entrada de Unity.

Contiene diferentes componentes que permiten las siguientes tareas de interacción:

- **Entrada de controladores XR multiplataforma** donde se incluyen las Meta Quest (anteriormente llamadas Oculus) y OpenXR entre otras.
- **Interacción** con objetos básicos donde se incluye colocar, seleccionar y agarrar.
- **Retroalimentación háptica**[1] a través de controladores XR.
- **Comentarios visuales** indicando así posibles interacciones.
- **Interacciones básicas** con la UI con los controladores XR.
- **Compatibilidad con XR Origin**, un equipo de cámaras de VR para controlar experiencias estacionarias.

Gracias a esta extensión se puede interactuar con los diferentes elementos del juego mediante los controladores XR además de poder incorporar XR Origin.

XR Device Simulator

XR Device Simulator [81] permite a los desarrolladores de proyectos con tecnologías XR probar de una forma rápida su proyecto sin necesidad de conectar dispositivos físicos de realidad virtual o aumentada.

Este componente maneja las entradas del ratón y del teclado y las utiliza para controlar los controladores XR de forma simulada junto con una pantalla en la cabeza (HMD) XR.

[1] A través del sentido del tacto.

De esta forma maneja indirectamente dichos controladores a través de dispositivos de entrada simulados. En este proyecto ha sido de gran utilidad debido a que no se disponía de dispositivos físicos de realidad virtual.

Meta XR Core SDK

El paquete Meta XR Core SDK [53] contiene activos y características básicas que ayudan a los desarrolladores a crear experiencias inmersivas para dispositivos Meta XR. Gracias a este paquete el proyecto podrá ser ejecutado en dichos dispositivos.

3.1.2. Unity Hub

Unity Hub es una herramienta muy útil de gestión centralizada para proyectos de Unity que se ha utilizado para [79]:

- Activar las licencias de Unity.
- Crear y gestionar diferentes proyectos facilitando así el seguimiento de múltiples proyectos y sus dependencias.
- Descargar y administrar diferentes versiones del Unity Editor permitiendo tener varios simultáneamente.
- Acceder a la Unity Assets Store donde se pueden descargar múltiples recursos para el desarrollo en Unity.

3.1.3. Visual Studio

Visual Studio es un entorno de desarrollo integrado (IDE) que se ha utilizado para la creación e implementación de los *scripts* que se usarán en Unity [60]. La principal razón por la que se ha elegido este IDE es por su excelente integración con Unity, además de su depuración avanzada y las extensiones y complementos que ofrece.

Figura 3.2: Logo de Visual Studio [2].

Las extensiones utilizadas han sido:

- **GitHub Copilot**: Herramienta de inteligencia artificial desarrollada por GitHub que funciona como un asistente de código para dar sugerencias sobre el código o autocompletarlo dependiendo del contexto. De esta forma se acelera el proceso de desarrollo.

- **CodeMaid VS2022**: Herramienta utilizada para organizar, limpiar y simplificar el código de C#.
- **SonarLint for Visual Studio 2022**: Herramienta que proporciona un análisis estático del código en tiempo real. Mejorando así la calidad del código.

C#

Como lenguaje de programación para la creación de los *scripts* y clases se ha elegido C# [80] debido a que es compatible con Unity y además es un lenguaje administrado, código cuya ejecución está administrada mediante un tiempo de ejecución. Una de sus ventajas es que la administración de memoria se hace de manera automática [65].

Figura 3.3: Logo de C# [51].

3.2. Tecnologías de comunicación y gestión del proyecto

Las tecnologías de comunicación y gestión del proyecto son aquellas que se han utilizado para facilitar la colaboración, coordinación y seguimiento del proyecto entre la alumna y los tutores.

3.2.1. Trello

Trello [76] es una herramienta visual que se ha utilizado para organizar las *issues* en cada *sprint* durante todo el proyecto. Está basado en el concepto de tablero, listas y tarjetas lo que lo hace muy intuitivo y fácil de usar.

Figura 3.4: Logo de Trello [102].

En este proyecto se ha creado un tablero que contiene 5 listas diferentes:

- **Tablero de tareas:** Aquí se encuentran todas las tareas relacionadas con la memoria del proyecto.
- **Tareas de desarrollo:** El resto de tareas del proyecto que pueden estar relacionadas con la implementación de funcionalidad o creación de diagramas.

- **En proceso:** Tareas que se deben hacer durante el *sprint*.
- **Validando:** Las tareas que ya han sido completadas pero que todavía tienen que ser validadas por los tutores.
- **Validado/Finalizado:** Tareas que ya han sido finalizadas y revisadas por los tutores.

3.2.2. Outlook

Outlook es un cliente de correo electrónico que permite enviar y recibir emails, administrar el calendario, almacenar nombres y números de los contactos y realizar un seguimiento de tareas [56]. Su primer lanzamiento fue en 1997 por Microsoft y forma parte de la suite de Microsoft Office [103].

Figura 3.5: Logo de Outlook [1].

En este proyecto se ha utilizado como principal forma de comunicación para organizar reuniones o preguntar dudas. También se ha utilizado el calendario para ver las reuniones de fin de *sprint*.

3.2.3. Zoom

Zoom [108] es una plataforma de videoconferencia que permite realizar reuniones con audio, vídeo y chat con varias personas a la vez. De esta forma se consigue una comunicación más eficaz y fluida.

Figura 3.6: Logo de Zoom [72].

Esta herramienta ha sido elegida por el tutor de HP-SCDS y ha sido utilizada en la gran mayoría de reuniones del proyecto.

3.2.4. Microsoft Teams

Microsoft Teams [57] es una aplicación de colaboración perfecta para utilizar en trabajos híbridos u online para que los equipos se mantengan informados, organizados y conectados. Entre sus principales funciones se encuentran el chat, *teams*, calendario y aplicaciones. En

nuestro caso ha sido utilizado como medida de contingencia en una ocasión en la cual Zoom no estaba operativo.

Figura 3.7: Logo de Microsoft Teams [100].

3.2.5. GitLab

GitLab [35] es una plataforma web creada en 2011 como un proyecto de código abierto cuyo objetivo es ayudar a los programadores a colaborar y entregar software de una forma eficiente y rápida a la vez que mejora la seguridad. Hace una gestión del ciclo del proyecto software mediante el control de versiones de código fuente, además de ser un repositorio online para almacenar proyectos.

Sus principales características son [25]:

- **Repositorio Git.** Mediante el control de versiones distribuido de Git puede gestionar y seguir los cambios en el código.
- **Gestión de proyectos.** Ampliamente utilizado para la gestión y organización de proyectos gracias a que permite crear y gestionar tareas, llevar un seguimiento...
- **Herramientas colaborativas.** Facilita la colaboración entre los miembros del equipo.
- **Integración Continua (CI) y Despliegue Continuo (CD).** Automatiza la construcción, las pruebas y el despliegue de aplicaciones.
- **Control de acceso y seguridad.** Permite configurar diferentes roles con diferentes niveles de acceso para poder controlar las acciones que se realizan dentro del repositorio.
- **Sistema eficiente de registros y seguimiento.** Permite llevar un seguimiento detallado del proyecto mediante registros de actividades, de los cambios y acciones realizadas en el repositorio.
- **Documentación de proyectos.** Permite crear y gestionar la documentación del proyecto.
- **Soporte para metodología *DevOps*.** Existen herramientas y funciones que permiten la implementación de prácticas y plataformas *DevOps* en el desarrollo de software.
- **Flexibilidad y escalabilidad.** GitLab puede usarse tanto en pequeños como grandes desarrollos, adaptándose a las necesidades de cada proyecto.

Esta herramienta es una de las impuestas por HP-SCDS, quien se ha encargado de la creación del repositorio del proyecto donde se ha desarrollado. Se ha utilizado como controlador de versiones, lo que permite revisar el código y tener un control sobre los avances y

Figura 3.8: Logo de GitLab [101].

mejoras. Esto facilita que los tutores puedan llevar un seguimiento del proyecto de forma eficiente y rápida.

Git

Git [31] es un sistema de control de versiones *Open Source* que se utiliza para el seguimiento de los cambios en el código de un proyecto. En proyectos donde colaboran varias personas facilita que puedan trabajar de forma simultánea sin problemas.

Figura 3.9: Logo de Git [30].

En Git, un repositorio es un sistema de almacenamiento de carpetas y ficheros que, gracias a los metadatos, se puede guardar a modo de histórico todos los cambios realizados. Una rama es un puntero a un estado del repositorio en un instante concreto y permite que un desarrollador pueda modificar esa rama de forma independiente y una vez haya terminado se hará un *merge* (operación que añade la funcionalidad a la rama principal).

En este proyecto se han utilizado cinco tipos de ramas de desarrollo:

- ***Main*:** también conocida como *Master*, es la rama principal del proyecto y la que contiene una versión del proyecto que se considera finalizada.
- ***Develop*:** rama de integración de las *features*. Una vez se llega a una versión del proyecto estable y con cambios significativos de la funcionalidad, se podrá mergear a *Main*.
- ***Feature*:** ramas creadas a partir de la rama *Develop* donde se desarrollará la nueva funcionalidad. Estas ramas comienzan con "Feature/" y continúa con una breve descripción de la funcionalidad a implementar. Una vez se mergean con *Develop* son eliminadas.
- ***Refactor*:** ramas creadas a partir de la rama *Develop* donde se realizará una refactorización del código. Estas ramas comienzan con "Refactor/" y al igual que en el apartado anterior sigue con una breve descripción de lo que se va a refactorizar. Una vez se mergea con *Develop* también es eliminada.
- ***BugFix*:** ramas que también son creadas a partir de *Develop* y que su objetivo es arreglar *bugs* que hayan surgido de alguna implementación anterior. Al igual que las de *feature* y *refactor*, una vez mergeada son eliminadas y en este caso comienzan con "BugFix/" y continúa con la descripción del *bug*.

3.3. Tecnologías de documentación y diseño

Las tecnologías de documentación son aquellas que han sido utilizadas para escribir la memoria del proyecto. Las tecnologías de diseño equivalen a las herramientas utilizadas para la creación de los diferentes diagramas que aparecen en la memoria.

3.3.1. Overleaf

Overleaf [77] es un editor de LaTeX colaborativo basado en la nube que permite escribir, editar y publicar documentos científicos. La memoria de este proyecto se ha realizado con este editor de LaTeX ya que permite la visualización instantánea del resultado del documento gracias a que el compilador está integrado. También es una opción excelente ya que al ser una plataforma web no necesita ninguna instalación ni actualizaciones, funciona en cualquier dispositivo y permite al tutor revisar el documento en cualquier momento. Además, también ha sido de utilidad el historial de las diferentes versiones del documento.

Figura 3.10: Logo de Overleaf [67].

3.3.2. Astah

Astah [6] es un *software* que contiene varias herramientas de modelado que permiten visualizar las ideas y diseños de *software* mediante la creación de diferentes diagramas. Las principales ventajas de Astah son su simplicidad, calidad y la comunidad que tiene. Se ha utilizado la versión *Astah Professional 9.0.0* para la creación de todos los diagramas de análisis y diseño del proyecto.

Figura 3.11: Logo de Astah Professional [5].

Capítulo 4

Requisitos de cliente

Desde la empresa HP-SCDS se han establecido una serie de requisitos que el estudiante debe cumplir. Los requisitos podrán ir cambiando a lo largo del proyecto para incluir o excluir funcionalidad. Esta lista es una explicación a alto nivel de lo que debería incluir la aplicación.

4.1. Requisitos de HP-SCDS

Se creará un videojuego en primera persona donde el usuario diabético es el protagonista absoluto. Una de las actividades que más se le pueden complicar a una persona diabética es la actividad física, ya que para ello ha de conocer su biorritmo a la perfección, sabiendo cómo le afecta la ingesta de hidratos, cuál es la hora donde más insulino-resistente es, además de cómo le va a afectar la propia actividad que está realizando en ese momento. Por ello, se propone ambientar el videojuego en una carrera tipo maratón. La mecánica será la siguiente:

- Como se ha comentado, estará desarrollado en primera persona, y se simulará que el usuario va por un circuito donde va corriendo. Por simplicidad, quizá se pueda partir de un circuito de atletismo.
- Se desarrollará en diferentes niveles, es un videojuego, por lo que debe ser jugable, y entretener, así que cada nivel será más complicado que el anterior. Posteriormente se hablará de la complejidad de cada nivel.
- Todos los niveles empezarán de la misma manera: el usuario elegirá cuántas raciones de hidratos ingerirá, cuánto rato esperará y cuántas unidades de insulina se aplicará. Esto dará un nivel de glucosa inicial sobre el que se comenzará la partida. Se presentarán varios alimentos a ingerir: de absorción rápida, moderada y lenta. En función de eso y el tiempo que decida esperar, se calculará el nivel de glucosa inicial. Más adelante se explicará cómo afecta cada alimento, el tiempo de espera y la insulina.

- Una vez calculado el nivel de glucosa inicial, comenzará la carrera. El jugador contará con una barra de energía que representará el nivel de glucosa en cada momento. Estará comprendido entre 0 y 500, siendo ambos límites extremadamente peligrosos, y el objetivo será mantenerlos entre 70 y 120, que son niveles de seguridad.
- Al ser un juego basado en una carrera con el fundamento del aprendizaje, en cada nivel la velocidad de la carrera irá aumentando. Inicialmente se parte de 5 niveles, y como extra, se propone un sexto donde el usuario sea quien decida la velocidad a la que ir.
- Al ser un circuito de atletismo, el usuario estará sólo, y en los diferentes carriles del circuito irán apareciendo elementos que modificarán su nivel de glucosa. Deberá decidir si los toma o los ignora en función de dicho nivel.
- Se plantea una carrera en el que, tras cada paso por meta, se incremente el nivel hasta llegar al final. Para hacerlo más emocionante, se irán sumando puntos a fin de que el juego sea más atractivo.

4.1.1. Cálculo del nivel de glucosa inicial

Para poder calcular el nivel de glucosa inicial, se debe conocer previamente cómo afecta cada elemento en el día a día de una persona diabética.

Primero se comenzará con las raciones de hidratos de carbono. Éstas son las que hace subir el nivel de glucosa en sangre a una persona con diabetes. Sin embargo, existen tres tipos de hidratos de carbono en función de su absorción:

- **Absorción lenta:** se encuentran en alimentos como legumbres, panes integrales, pastas y arroces al dente. Una ración de hidratos de carbono equivale a 10 gramos de un hidrato específico. Por tanto, se considerará que una ración de este tipo de alimentos incrementará en 100 nuestro nivel de glucosa en sangre en un periodo de una hora.
- **Absorción moderada:** Las frutas y la leche se consideran de absorción moderada. Una ración de hidratos incrementará en 100 el nivel de glucosa en sangre en un periodo de 30 minutos. La bollería podría encajar aquí también.
- **Absorción rápida:** Azúcar, miel, caramelos, zumos... todo ello se considera de absorción rápida y aportarán 100 unidades extra al nivel de glucosa en apenas 15 minutos por cada ración de hidratos ingerida.

Se ha hablado de qué sucede cuando se ingieren hidratos de carbono en un determinado periodo de tiempo, pero para evitar esos picos de glucemia, se suele aplicar insulina (explicada en la Sección 1.1).

Para el videojuego, se utilizará únicamente la insulina de acción semi-rápida, ya que la de acción lenta se entiende que ya se habrá aplicado, y la de acción rápida es para casos especiales. Por tanto, para llevar a cabo el cálculo inicial para cada nivel, se parte de unas medidas estándar. Para una ración de hidratos de acción lenta, con un reposo de una hora,

y una aplicación de 10 unidades de insulina, el nivel de glucosa se incrementará únicamente en 10 unidades, en lugar de 100. Es decir, al aplicar insulina, se contrarresta el efecto de los hidratos de carbono.

4.1.2. Interferencia de elementos en la pista

Ya se ha hablado de cómo afectan las raciones de hidratos y de insulina, así que ahora se verá cómo afectarán los elementos que irán apareciendo en la pista.

Una vez ha comenzado el nivel, se ha indicado que la velocidad de nuestro personaje irá aumentando por niveles. Como es una actividad física, ante la ausencia de hidratos y de insulina, el nivel de glucosa irá descendiendo a una velocidad de 2 unidades por minuto. Como hacer una carrera que dure mucho puede hacer que el juego sea aburrido, se va a suponer que cada vuelta dura en torno a 1:30 a 2:00 minutos, por lo que el nivel de glucosa descenderá en 2 unidades cada 2 segundos.

Por ejemplo, si se inicia el juego con 180 unidades de glucosa, a 2 minutos, al final de la carrera estará en 60 unidades de glucosa (lo que ya es una hipoglucemia), ya que en 2 minutos habrá bajado el nivel en 120 unidades.

Al igual que con el ejercicio físico, también es necesario adaptar cómo afectan el resto de elementos con respecto al transcurso del tiempo:

- **Alimentos con hidratos de carbono de absorción rápida:** incrementarán el nivel de glucosa en 4 unidades cada 2 segundos durante 20 segundos.
- **Alimentos con hidratos de carbono de absorción moderada:** incrementarán el nivel de glucosa en 3 unidades cada 2 segundos durante 20 segundos.
- **Alimentos con hidratos de carbono de absorción lenta:** incrementarán el nivel de glucosa en 2 unidad cada 2 segundos.
- **Insulina:** reducirá el nivel de glucosa en 4 unidades cada 2 segundos.
- **Actividad física:** reducirá el nivel de glucosa en 2 unidades cada 2 segundos.

Si el usuario decide tomar alguno de estos elementos, se sumará o restará al de la actividad física como corresponda y serán acumulativos.

4.1.3. Suma de puntos

La suma de puntos será de la siguiente manera:

- Cada nivel empieza con 1000 puntos por defecto.

- Cada vez que supere el nivel de 120 unidades, restará 5 puntos e irá restando 1 punto por cada segundo que esté por encima de ese valor.
- Cada vez que baje del nivel 70, restará 10 puntos e irá restando 2 puntos por cada segundo por debajo de ese nivel.
- En cuando a los obstáculos durante la carrera, la fruta suma 5 puntos, el zumo resta 10 puntos, el pan suma 10 puntos, la bollería resta 20 puntos y la carne no suma ni resta.
- La insulina no suma ni resta.
- Al llegar a cada checkpoint, si el nivel de glucosa está entre 70 y 120, no suma ni resta, si está entre 120 y 180, resta 200, si está por encima de 180 resta 300, y si está por debajo de 70, resta 400.
- Si el jugador llega al nivel 0 de glucosa, termina la maratón y va al hospital.
- Si el jugador está por encima de 300 durante 5 segundos, termina la maratón y se tiene que quedar quieto y aplicar insulina.

El reto para el jugador es terminar la maratón con la máxima puntuación y el objetivo es enseñar cómo afecta la actividad física, la alimentación y la insulina a sus niveles de glucosa.

Capítulo 5

Planificación y gestión del proyecto

5.1. Scrum

En este proyecto se ha seleccionado **Scrum** como marco de gestión de proyectos de metodología ágil. Scrum [21] anima a los equipos a aprender mediante la experiencia, a poder autogestionarse cuando aparece un problema y a reflexionar sobre las victorias y derrotas para seguir mejorando. Se compone de un equipo pequeño que será el encargado de desarrollar, entregar y mantener sus productos utilizando un enfoque iterativo e incremental en pequeños fragmentos de tiempo.

Valores de Scrum

Los valores de Scrum [21] se centran en el trabajo, las acciones y la conducta de los integrantes del equipo de Scrum. En 2016, la Guía de la metodología Scrum añadió cinco valores que se consideran esenciales para el éxito de los equipos:

Compromiso Los equipos de Scrum son pequeños por lo que todos los miembros desempeñan un papel muy importante. Por ello todos deben estar comprometidos a completar las tareas que tengan asignadas.

Valor Valor de probar cosas nuevas y de cuestionar todo aquello que no permita llegar a los objetivos establecidos. Valor para ser transparente a la hora de encontrarse con obstáculos, de hablar sobre el progreso del proyecto, los retrasos...

Foco Scrum se centra en *sprints*, período de tiempo acotado y centrado en que el equipo cumpla unas determinadas tareas, explicado más detenidamente más adelante. Es la forma de estructurar y de ayudar a completar todas las tareas planificadas.

Honestidad Se realizan unas reuniones diarias donde se promueve la transparencia al hablar del trabajo realizado y de los contratiempos que hayan podido surgir.

Respeto Dentro del equipo es muy importante colaborar y reconocer las contribuciones de cada miembro del equipo al *sprint*. Se deben celebrar los logros de los demás y se deben respetar unos a otros.

Miembros de un equipo en Scrum

Como ya se ha explicado los equipos de Scrum son reducidos, no más de 10 personas. Estos equipos deben tener 3 roles específicos [21]:

- ***Product Owner*:** Es la persona que más conoce el producto y su objetivo es entender los requisitos del cliente y del mercado para luego poder priorizar el trabajo. Encargado de crear y gestionar el *backlog* del producto, término que se explicará en el apartado de artefactos. Además, aportan al equipo instrucciones claras sobre las tareas que deben entregar y deciden cuándo se debe lanzar el producto.

- ***Scrum Master*:** Es la persona que conoce muy bien el trabajo que realiza el equipo y que puede ayudar a mejorar su transparencia y flujo de entrega. También se encarga de planificar los recursos necesarios para organizar los diferentes eventos de Scrum.

- **Equipo de desarrollo:** Son las personas que se encargan de realizar el desarrollo de las tareas. Tienen habilidades muy variadas y se forman entre ellos para que nadie suponga un cuello de botella a la hora de entregar las tareas. Por ello todos los miembros del equipo se ayudan entre sí para conseguir finalizar el *sprint* con éxito.

La Figura 5.1 refleja estos roles y las interacciones que realizan.

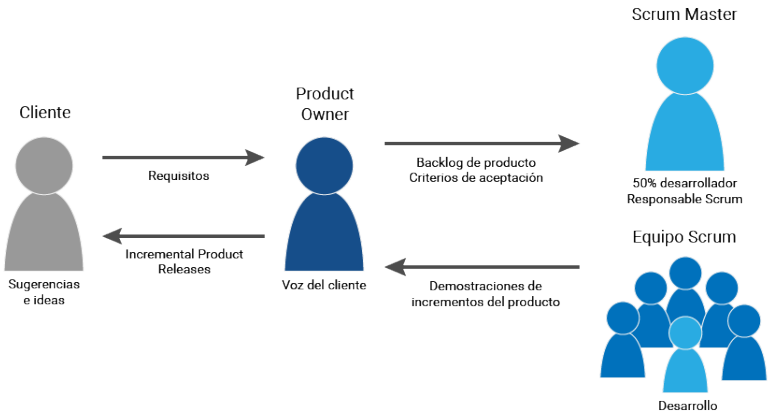

Figura 5.1: Diferentes roles dentro del equipo de Scrum [18]

Artefactos de Scrum

Los artefactos de Scrum [21] ayudan al equipo a recopilar información importante para definir el producto y el trabajo que se debe realizar para crearlo. Existen tres tipos de artefactos:

- ***Product Backlog*:** Lista de tareas que se deben realizar, ya sea nuevas funcionalidades, mejoras o correcciones. El *Product Owner* se encarga de revisar, cambiar prioridades y mantener el *backlog* del producto actualizado, ya que a medida que se avanza en el desarrollo pueden encontrarse con nuevas tareas que haya que realizar.
- ***Sprint Backlog*:** Lista de tareas que se han seleccionado del *Product Backlog* para implementarlas en ese *sprint*. Antes de comenzar el *sprint*, el equipo elige que tareas va a realizar. El *Sprint Backlog* puede ser flexible siempre y cuando no se ponga en riesgo el objetivo principal del *sprint*, lo que el equipo quiere lograr en ese *sprint*.
- **Incremento (Objetivo del *sprint*):** es el producto final utilizable del *sprint*. Al final del *sprint* se realiza una demostración donde el equipo muestra lo que se ha completado en dicho *sprint*.

Eventos de Scrum

En Scrum [21] se incluyen diferentes prácticas, protocolos y reuniones que los equipos realizan de forma regular. En esta parte es donde vamos a ver más variaciones ya que pueden existir equipos que piensen que realizar todos los protocolos es engorroso y otros que lo vean como una parte fundamental para el correcto desarrollo del producto. Lo más recomendable es probar usando todos los protocolos y luego hacer una retrospectiva rápida para ver qué es lo que mejor funciona en el equipo.

- **Organización del *backlog*:** Se encarga el *Product Owner* y consiste en realizar el mantenimiento de esta lista de tareas mediante las opiniones de usuarios y del equipo de desarrollo para así poder priorizar y mantener limpia la lista para poder trabajar sobre ella en cualquier momento.
- ***Sprint Planning*:** Es la reunión inicial del *sprint* donde se reúne todo el equipo de desarrollo y el *Scrum Master* para planificar que tareas se van a realizar durante el *sprint* actual y establecer su objetivo. El *Scrum Master* es el encargado de dirigir esta reunión
- ***Sprint*:** Periodo de tiempo donde el equipo de desarrollo trabaja para completar las tareas que se encuentran en el *Sprint Backlog*. La duración de estos suelen ser de dos semanas pero esto dependerá de cada equipo e incluso de la complejidad de las tareas. Cuanto más complejo y cuanta más incertidumbre exista, los *sprints* deberían ser más cortos. Este periodo es flexible y el alcance se puede modificar en cualquier momento si se ve necesario.

  En estos periodos de tiempo acotados también tienen lugar todos los eventos de Scrum.

- ***Daily Scrum*:** Reunión diaria de unos 10-15 minutos que se realiza siempre a la misma hora del día. A las *Dailys Scrum* también se las denomina reunión rápida y es que es muy importante que sean rápidas. El objetivo es coordinar a todo el equipo para que estén en sintonía, también es el momento perfecto para expresar todas las inquietudes sobre el objetivo del *sprint* o de comentar los problemas que hayan surgido.

  Lo más habitual es que cada integrante del equipo responda estas tres preguntas:

  - ¿Qué hice ayer?
  - ¿Qué voy a hacer hoy?
  - ¿Tengo algún impedimento con mis tareas?

- ***Sprint Review*:** Cuando finaliza el *sprint*, el equipo se reúne para hacer una demostración del trabajo realizado a lo largo de ese *sprint* y se analizan los resultados del *sprint*. Se muestran las tareas que han sido completadas a las partes interesadas y al resto de integrantes del equipo para recibir comentarios. Aquí el *Product Owner* puede decidir si quiere publicar el incremento o no y se hace una revisión del *Product Backlog* para tenerlo preparado en la planificación del siguiente *sprint*.

- ***Sprint Retrospective*:** Esta reunión sirve para documentar y analizar los aspectos que han funcionado y que no han funcionado de un *sprint*, del proyecto, de herramientas, protocolos o incluso de las personas. Se trata principalmente de dedicar un tiempo a reflexionar sobre qué cosas han salido bien y qué puede mejorar para la próxima vez, aunque también es importante identificar qué salió mal para poder cambiarlo.

En la Figura 5.2 se puede apreciar todo el ciclo del que se compone Scrum.

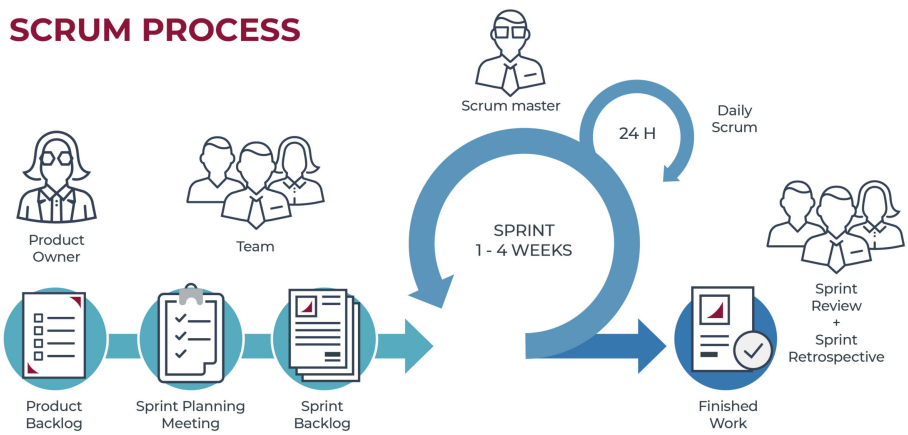

Figura 5.2: Eventos y artefactos en Scrum [17]

5.1.1. Adaptación de Scrum a SugarRace

El principal motivo por el que se ha elegido Scrum y no un marco de trabajo basado en una metodología tradicional como puede ser cascada es porque **Scrum es ágil**, es decir, se adapta fácilmente a cualquier cambio y aunque en este proyecto se parta de unos requisitos dados por la empresa HP-SCDS, estos están abiertos a cambios, pudiendo añadir o modificar si se ve conveniente. Además, aparte de la **flexibilidad y adaptabilidad** que proporciona, la **entrega de incrementos** ayuda a que los tutores puedan dar *feedback* del trabajo que se está realizando.

Otra parte importante por la que se ha elegido Scrum y no otra metodología ágil como Kanbas, es porque tiene unos **eventos bien definidos** como el *Sprint Planning* o el *Sprint Review*.

Sin embargo, para poder aplicar este marco de trabajo en SugarRace se ha tenido que realizar ciertas adaptaciones para poder usarlo en el trabajo de fin de grado ya que Scrum es ampliamente utilizado en el ámbito laboral y no está tan enfocado a este tipo de proyectos.

En primer lugar, el equipo se compone del tutor académico de la UVa, Cesar Pablo Gutiérrez Martínez, del tutor de la empresa HP-SCDS, Luis Vidal de la Rosa, y de la alumna, María José García Matías. Los roles, explicados anteriormente, se han adaptado. **Los tutores asumen el rol de *Product Owner*** por lo que priorizan las tareas del *Product Backlog*, proporcionan claridad y dan soporte para posibles dudas sobre el producto, participan en la planificación y revisión de *sprint* y revisan los incrementos de cada *sprint*.

El alumno asume tanto el rol de *Scrum Master* como el de Equipo de Desarrollo. Al ser el *Scrum Master* es el encargado de organizar y facilitar los eventos de Scrum además de revisar de que el *sprint* se desarrolle de forma correcta. Además, también deberá desarrollar todo el proyecto por asumir el rol de Equipo de Desarrollo. Al ser un TFG, la responsabilidad de crear las tareas del *Product Backlog* también han sido delegadas a la alumna aunque esta sería una tarea del *Product Owner*.

La adaptación de los eventos de Scrum se ha realizado de manera que tanto la reunión inicial, *Sprint Planning*, como la reunión de fin de *sprint*, *Sprint Review*, y la retrospectiva, *Sprint Retrospective*, se hagan **los jueves de cada 3 semanas en una única reunión** donde se analizará el trabajo realizado durante el *sprint* que ha finalizado, se hará una retrospectiva destacando qué ha ido bien, qué se puede mejorar y qué ha ido mal, y por último se planificará el siguiente *sprint*. En estas reuniones también estarán presentes los tutores. En cuanto a la *Daily Scrum* se llevará a cabo por el estudiante antes de ponerse a trabajar. En ellas se deberá hacer un auto-análisis de lo que se hizo en el día anterior y de las tareas que se van a realizar ese mismo día.

5.2. Planificación y estimación inicial

La planificación inicial se basará en *sprints*. Como ya se ha explicado en la sección anterior, la duración de los *sprints* será de 3 semanas excepto el Sprint 0, denominado *Kick-off*. Este

Sprint 0 se centra en la preparación y organización del proyecto. Se tendrá un primer contacto con las nuevas tecnologías que se van a utilizar en el trabajo de fin de grado como Unity o Visual Studio. También se analizarán los requisitos junto con una pequeña investigación sobre la arquitectura del proyecto y se crearán todas las historias de usuario que formarán parte del *Product Backlog* inicial partiendo de la idea original del proyecto. Como ya se ha explicado anteriormente el *Product Backlog* es flexible por lo que durante el desarrollo del proyecto se podrán modificar y añadir historias de usuario.

En la Figura 5.3 podemos observar los diferentes *sprints* que se van a realizar junto con las fecha de inicio y fin de cada uno. El Sprint 0 tiene una duración de más de dos meses ya que no se dio comienzo hasta que no terminaron los exámenes ordinarios y extraordinarios del primer cuatrimestre. Los 6 *sprints* restantes tendrán una duración de 3 semanas y la fecha de inicio de cada uno corresponde con una reunión con los tutores, donde se llevarán a cabo los eventos de Scrum como ya se ha mencionado en la Sección 5.1.1.

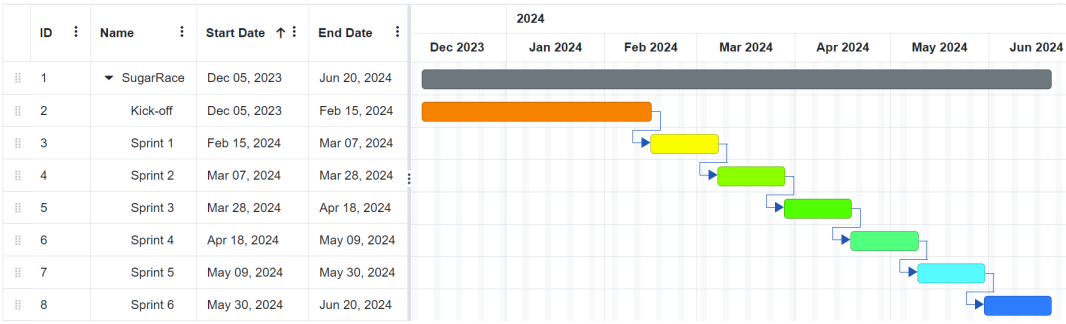

Figura 5.3: Diagrama de Gantt del proyecto

Para elaborar el diagrama de Gantt de la Figura 5.3 y elegir la fecha de inicio se han tenido en cuenta la fecha límite para la entrega en convocatoria ordinaria del trabajo de fin de grado, el 20 de Junio de 2024, y el tiempo que se debe dedicar al mismo, que según la guía docente [95] tiene una carga de *12 ECTS*, es decir, 300 horas.

5.2.1. Estimación inicial

Teniendo en cuenta el diagrama de Gantt de la Figura 5.3 donde ya está definida la fecha de inicio, fin y cada una de las fechas de los *sprints*, ya se podría pasar a la estimación inicial de las horas que se planea trabajar durante cada *sprint* teniendo en cuenta las diferentes circunstancias (exámenes, vacaciones...).

Si el proyecto consta de 7 *sprints* y tiene una carga de 300 horas, habría que trabajar 43 horas aproximadamente en cada *sprint*. Como ya se ha explicado anteriormente, no se trabajarán las mismas horas en todos los *sprints* y por ello se ha creado la Tabla 5.1 donde se encuentra una estimación inicial orientativa de cada uno de los *sprints*. Posteriormente, en el Capítulo 10 se estimarán los diferentes sprints teniendo en cuenta dicha tabla pero podría variar ya que la estimación se realizará por cada una de las tareas.

Fase	Horas	Justificación
Kick-off	45 horas	Primer cuatrimestre y exámenes ordinarios
Sprint 1	45 horas	
Sprint 2	45 horas	
Sprint 3	35 horas	Prácticas y exámenes
Sprint 4	45 horas	
Sprint 5	45 horas	
Sprint 6	40 horas	
Total	**300 horas**	

Tabla 5.1: Estimación inicial

5.3. *Product Backlog*

En esta sección se verá el *Backlog* del producto creado a partir del análisis de los requisitos del cliente y de los conocimientos e ideas iniciales del proyecto. Tal y como se ha explicado en la sección anterior el *Product Owner* tiene la responsabilidad de crear las historias de usuario en el *Product Backlog* pero al ser un TFG esta responsabilidad ha sido delegada a la alumna. El *Backlog* recogerá todas las historias de usuario que se deben realizar para completar el proyecto. Se añadirán y modificarán las historias de usuario según se crea conveniente durante el desarrollo.

Las historias de usuario (HU) seguirán la siguiente estructura:

- **Título:** Breve descripción del objetivo o tarea que se va a realizar.
- **Descripción (HU):** Describe la funcionalidad o requisito desde el punto de vista del usuario. Sigue la siguiente estructura:
  - **Como** <usuario, desarrollador, otra persona>
  - **Quiero** <breve explicación sobre lo que la persona especificada en el 'Como' necesita>
  - **Para** <por qué eso será útil para esa persona>
- **Criterios de aceptación:** Mínimo que se debe cumplir para que la HU se considere como completada. Normalmente se usan afirmaciones claras y específicas sobre las condiciones que se deben cumplir para que no haya malentendidos.
- ***Label*:** Son etiquetas que se asignan a cada historia de usuario para poder clasificar y ordenar mejor todas las funcionalidades que se deben desarrollar. Los *labels* que se han utilizado son:
  - ***Development*:** HU relacionada con el desarrollo de software.
  - ***Spike*:** HU relacionada con la investigación de un problema técnico para reducir la incertidumbre [54].
  - ***Enviroment*:** HU relacionada con el diseño del entorno donde se desarrollará el juego junto con la funcionalidad.

- ***Configuration*:** HU relacionada con la configuración del entorno de desarrollo.
- ***Algorithm*:** HU relacionada con la implementación de algoritmos específicos.

HU1	
Título	Configuración del entorno de desarrollo.
Descripción (HU)	**Como** desarrollador, **Quiero** configurar el entorno de desarrollo del proyecto, **Para poder** implementar las funcionalidades que se especifican en los requisitos.
Criterios de aceptación	▪ Instalación de Unity Hub, Unity Editor y Visual Studio 2022. ▪ Instalación de los *plugins* de Visual Studio: CodeMaid, SonarLint, GitHub Copilot. ▪ Creación de un proyecto 3D.
Label	*Configuration*

Tabla 5.2: Historia de usuario 1

HU2	
Título	Encontrar recursos para simular una pista de atletismo o similar en Unity.
Descripción (HU)	**Como** desarrollador, **Quiero** encontrar recursos para el entorno del juego, **Para poder** crear un entorno inmersivo para el usuario.
Criterios de aceptación	▪ Los recursos deben ser en 3 dimensiones. ▪ Los recursos deben ser compatibles con Unity. ▪ Los recursos deben ser adquiridos de una fuente fiable y de calidad. ▪ Los recursos deben poder ser personalizables para que se ajuste a las necesidades del proyecto. ▪ Los recursos deben ser legales y se deben respetar los derechos de autor y de propiedad intelectual. ▪ Los recursos deben adaptarse al modo de juego y público objetivo del proyecto. ▪ Los recursos deben ser visualmente atractivos.
Label	*Spike*, *Enviroment*

Tabla 5.3: Historia de usuario 2

HU3	
Título	Implementación de los recursos que simulan el entorno y el avatar del juego.
Descripción (HU)	**Como** desarrollador, **Quiero** realizar una composición de los recursos junto con el avatar, **Para poder** simular el entorno del juego.
Criterios de aceptación	▪ El entorno debe ser coherente, es decir, no debe haber inconsistencias como zonas mal construidas con huecos o vacíos. ▪ El entorno debe visualizarse correctamente. ▪ El entorno debe tener cielo definido en línea con los recursos elegidos. ▪ El entorno debe ser compatible con el resto de elementos que se añadan posteriormente, como los alimentos o la insulina. ▪ El avatar debe visualizarse correctamente representando a una persona corriendo.
Label	*Enviroment*

Tabla 5.4: Historia de usuario 3

HU4	
Título	Permitir movimientos del usuario.
Descripción (HU)	**Como** usuario, **Quiero** moverme por el entorno, **Para poder** coger o evitar objetos.
Criterios de aceptación	▪ El usuario únicamente tendrá la capacidad de moverse en el eje horizontal durante la carrera, es decir, no podrá avanzar, retroceder, saltar, ni realizar cualquier otro movimiento. ▪ El usuario solo se podrá mover de forma discreta para evitar que se quede entre carriles. ▪ El movimiento no debe exceder los límites del entorno.
Label	*Development*

Tabla 5.5: Historia de usuario 4

HU5	
Título	Creación del menú principal y de la pantalla de fin de nivel.
Descripción (HU)	**Como** usuario, **Quiero** visualizar el menú principal y la pantalla de fin de nivel, **Para poder** realizar las acciones disponibles en cada menú.
Criterios de aceptación	▪ El menú principal debe permitir la selección de nivel e informar al usuario de los objetos y del efecto que tienen en cada nivel. ▪ La pantalla de fin de nivel debe mostrar información diferente dependiendo de si ha superado el nivel o no. ▪ Se debe permitir jugar o salir del juego. ▪ Deben ser adecuados para el público objetivo. ▪ Deben ser coherentes y consistentes con la temática del juego. ▪ Deben ser usables.
Label	*Development*, *Enviroment*

Tabla 5.6: Historia de usuario 5

HU6	
Título	Encontrar recursos para simular alimentos e insulina en Unity.
Descripción (HU)	**Como** desarrollador, **Quiero** encontrar recursos 3D, **Para poder** representar alimentos e insulina.
Criterios de aceptación	▪ Los recursos deben ser en 3 dimensiones. ▪ Los recursos deben ser compatibles con Unity. ▪ Los recursos deben ser adquiridos de una fuente fiable y de calidad. ▪ Los recursos deben poder ser personalizables para que se ajuste a las necesidades del proyecto. ▪ Los recursos deben ser legales y se deben respetar los derechos de autor y de propiedad intelectual. ▪ Los recursos deben adaptarse al modo de juego y público objetivo del proyecto. ▪ Los recursos deben ser visualmente atractivos.
Label	*Spike*, *Enviroment*

Tabla 5.7: Historia de usuario 6

HU7	
Título	Generación del entorno y de los diferentes objetos del juego.
Descripción (HU)	**Como** usuario, **Quiero** visualizar el entorno y los diferentes objetos en él, **Para poder** obtener una buena experiencia de juego.
Criterios de aceptación	▪ El *script* debe ser general para poder utilizarlo siempre que haya que generar elementos. ▪ El *script* debe ser parametrizable, pudiendo modificar: • Los recursos. • La velocidad a la que se mueven. • La distancia entre uno y otro. • Un desplazamiento. • La dirección en la que se tienen que generar. ▪ No se pueden generar objetos en la misma posición donde aparece el usuario al inicio del nivel. ▪ Los objetos no pueden coincidir en la misma posición. ▪ Todo los elementos que se generen deben moverse hacia el jugador para simular que se está moviendo él y no el entorno. ▪ El circuito debe moverse a una velocidad adecuada, permitiendo al usuario un tiempo para poder tomar decisiones. ▪ Los objetos deben moverse a la misma velocidad que el circuito. ▪ En cada nivel se debe poder elegir que elementos deben aparecer.
Label	*Development*, *Enviroment*

Tabla 5.8: Historia de usuario 7

HU8	
Título	Recolección de alimentos e insulina.
Descripción (HU)	**Como** usuario, **Quiero** coger objetos, **Para poder** simular la ingesta de alimentos o la aplicación de insulina.
Criterios de aceptación	▪ El usuario debe ser capaz de coger los objetos que estén en su carril. ▪ Una vez el usuario coge el objeto debe desaparecer del entorno. ▪ En el *script* al coger cualquier elemento debe enviar un evento para posteriormente responder en consecuencia.
Label	*Development*

Tabla 5.9: Historia de usuario 8

HU9	
Título	Creación de la barra que simula el nivel de glucosa.
Descripción (HU)	**Como** usuario, **Quiero** saber mi nivel de glucosa, **Para poder** mantenerlo en un nivel saludable.
Criterios de aceptación	▪ La barra de glucosa debe aparecer en la parte superior derecha del campo de visión del usuario. ▪ La barra de glucosa debe tener un *script* asociado donde capte los eventos de recolectar objetos. ▪ El nivel de glucosa al ingerir alimentos debe aumentar y al aplicar insulina disminuir, dependiendo del nivel se utilizará un algoritmo diferente. ▪ La barra de glucosa debe abarcar el rango 0-300. ▪ Si el usuario tiene el nivel de glucosa entre 70 y 120 la barra de glucosa debe ser verde. ▪ Si el usuario tiene el nivel de glucosa por debajo de 70 o mayor que 120 la barra de glucosa debe ser roja. ▪ Si el usuario llega a un nivel de glucosa de 0 o de 300 se debe terminar el nivel. ▪ El nivel de glucosa estará representado por una barra y un valor numérico que se actualizarán en tiempo real.
Label	*Spike*, *Development*

Tabla 5.10: Historia de usuario 9

HU10	
Título	Cálculo de puntos.
Descripción (HU)	**Como** usuario, **Quiero** ganar o perder puntos, **Para** tener una experiencia de juego más divertida.
Criterios de aceptación	▪ Se debe comenzar el nivel con 1000 puntos. ▪ Se deben restar 5 puntos cada vez que el usuario supere el nivel de 120 unidades e irá restando 1 punto por cada segundo que esté por encima de ese valor. ▪ Se deben restar 10 puntos cada vez que el usuario baje del nivel de 70 unidades e irá restando 2 puntos por cada segundo que esté por debajo de ese valor. ▪ Se deben sumar 100 puntos cada vez que el usuario recoja un objeto y mantenga su nivel de glucosa en el rango correcto o se esté acercando a él. ▪ Se deben restar 200 puntos cada vez que el usuario recoja un objeto y el nivel de glucosa salga del rango correcto o se esté alejando de él. ▪ Se deben restar 200 puntos si el usuario llega a un checkpoint con el nivel de glucosa en el rango 120-180. ▪ Se deben restar 300 puntos si el usuario llega a un checkpoint con el nivel de glucosa por encima de 180. ▪ Se deben restar 400 puntos si el usuario llega a un checkpoint con el nivel de glucosa por debajo de 70. ▪ No se debe ni sumar ni restar puntos si el usuario llega a un checkpoint con el nivel de glucosa en el rango 70-120. ▪ Debe aparecer el valor numérico en la parte superior izquierda de la visión del usuario y actualizarse en tiempo real.
Label	*Algorithm, Development*

Tabla 5.11: Historia de usuario 10

HU11	
Título	Algoritmos del cálculo de nivel de glucosa
Descripción (HU)	**Como** desarrollador, **Quiero** crear un algoritmo diferente para cada nivel sobre el cálculo del nivel de glucosa, **Para poder** enseñar diferentes conceptos sobre la diabetes en cada nivel.
Criterios de aceptación	▪ El algoritmo del nivel 1 debe ser el más sencillo y la complejidad irá aumentando a medida que se suba de nivel. ▪ En el nivel 1 el efecto de tomar alimentos y de aplicar insulina será inmediato, sumando 20 al nivel de glucosa si ingiere comida o restando 20 si se aplica insulina. ▪ El nivel 2 es igual al 1 pero cada 2 segundos se restarán 2 unidades al nivel de glucosa por el ejercicio físico. ▪ En el nivel 3 se implementará un alimento de cada tipo de absorción: • **Absorción rápida:** Debe sumar 4 unidades cada 2 segundos durante 20 segundos. • **Absorción moderada:** Debe sumar 3 unidades cada 2 segundos durante 20 segundos. • **Absorción lenta:** Debe sumar 2 unidades cada 2 segundos hasta el final de la carrera. • **Libre absorción:** No sumará ni restará. La insulina restará 3 unidades cada 2 segundos durante 20 segundos. ▪ El nivel 4 debe ser igual al 3. ▪ En el nivel 5 se implementarán diferentes tipos de alimentos para cada tipo de absorción. En cuanto a lo que suma y resta cada uno será igual al nivel 3.
Label	*Algorithm*, *Development*

Tabla 5.12: Historia de usuario 11

HU12	
Título	Implementación de la decisión inicial del usuario.
Descripción (HU)	**Como** usuario, **Quiero** decidir cuántas raciones de hidratos ingerir, cuánto tiempo esperar y cuántas unidades de insulina aplicar, **Para poder** calcular mi nivel de glucosa inicial del juego.
Criterios de aceptación	▪ Se debe poder elegir entre tres tipos de alimentos: absorción lenta, rápida y moderada. ▪ Se debe poder elegir un tiempo que simulará lo que el usuario espera antes de comenzar el maratón. ▪ Se debe poder elegir el número de unidades de insulina que se deben aplicar. ▪ Absorción lenta: una ración de hidratos incrementa 100 unidades el nivel de glucosa en un periodo de una hora. ▪ Absorción moderada: una ración incrementa 100 unidades en un periodo de media hora. ▪ Absorción rápida: Una ración incrementa 100 unidades en un periodo de 15 minutos. ▪ Insulina: resta 10 unidades en un periodo de una hora. ▪ En los niveles 1 y 2 el nivel de glucosa del usuario será de 90 unidades y dependiendo de las decisiones se recalculará en tiempo real. ▪ En el nivel 3 el nivel de glucosa será aleatorio, entre 50 y 200, y dependiendo de las decisiones se recalculará en tiempo real. ▪ En el nivel 4 y 5 el nivel de glucosa será aleatorio como en el nivel 3 pero no se recalculará.
Label	*Algorithm*, *Development*

Tabla 5.13: Historia de usuario 12

HU13	
Título	Implementación de la realidad virtual.
Descripción (HU)	**Como** usuario, **Quiero** sumergirme en una experiencia orientada al metaverso y en realidad virtual durante el juego, **Para poder** aprender sobre la diabetes de una forma interactiva y realista.
Criterios de aceptación	▪ Se debe instalar la extensión XR Plugin Management, junto con el *Plugin Provider OpenXR* y el *Mock HMD Loader*. ▪ Se debe instalar y configurar el paquete XR Interaction Toolkit y *XR Origin*. ▪ Se debe de añadir un simulador de realidad virtual como XR Device Simulator. ▪ Se debe de permitir que el usuario pueda visualizar todo el entorno en los menús pero no podrá moverse. ▪ Durante la carrera se permitirá mover al usuario en el eje horizontal mediante el controlador izquierdo de realidad virtual y únicamente podrá mirar hacia delante. ▪ Debe ser compatible con el metaverso.
Label	*Development*

Tabla 5.14: Historia de usuario 13

5.4. Recursos del proyecto

Una vez conocemos el marco de trabajo que se va a utilizar y la planificación inicial del proyecto, ya podemos identificar los recursos que se van a necesitar para desarrollar el proyecto. Antes de comenzar, se explicará qué es un recurso y cómo se clasifican para luego concretar los que son necesarios para poder realizar este trabajo de fin de grado.

5.4.1. ¿Qué es un recurso?

Un **recurso** es cualquier elemento o persona necesarios para la realización de un proyecto [106]. Esta definición del libro *Software project management* es bastante amplia, desde un lápiz hasta las personas serían recursos. En este caso no llegaremos a tanto detalle y nos centraremos en los más relevantes dentro de este proyecto.

Otra cuestión importante es cuándo será necesario utilizar un recurso específico y durante cuánto tiempo será utilizado. Un recurso podría ser utilizado en una única tarea y no sería necesario mantenerlo durante todo el transcurso del proyecto.

Los recursos pueden clasificarse en diversas categorías, cada una de las cuales desempeña un papel crucial en el proceso de desarrollo. A continuación, se presentan siete categorías junto con una breve descripción de cada una [106]:

- **Mano de obra:** Miembros del equipo del proyecto de desarrollo, como los desarrolladores software.
- **Equipamiento:** Artículos de oficina, equipos informáticos y puestos de trabajo.
- **Materiales:** En este caso son artículos que se consumen, no que se utilizan. Normalmente en la mayoría de proyectos de software no son relevantes.
- **Espacio:** Entorno físico necesario para llevar a cabo el desarrollo del proyecto.
- **Servicios:** Contratación de servicios externos que pueden ser necesarios durante el desarrollo del proyecto. Incluye licencias para el uso de un software determinado.
- **Tiempo:** Es un recurso que se compensa con el resto de recursos primarios: Si aumentas el resto de recursos en un proyecto, los plazos pueden reducirse.
- **Dinero:** Es un recurso secundario utilizado para comprar el resto de recursos y que se consume a medida que se utilicen los recursos.

5.4.2. Plan de asignación de recursos

El primer paso para crear un plan de asignación de recursos es crear una enumeración de los recursos que se van a necesitar. Para ello, se examinarán las diferentes historias de usuario del *Product Backlog* inicial y se identificarán los recursos necesarios para cada una de ellas. Sin embargo, es común que existan recursos que no son específicos de una actividad, sino que forman parte de la infraestructura del proyecto o son necesarios para apoyar a otros recursos.

El segundo paso consistiría en evaluar la distribución de los recursos necesarios a lo largo del proyecto. En este caso, todos los recursos que se mencionan serán utilizados durante todo el desarrollo del proyecto.

Recursos necesarios

Los recursos necesarios para este trabajo de fin de grado los clasificaremos según las categorías explicadas anteriormente:

Mano de obra. El desarrollo del proyecto se atribuye a un único de desarrollador.

Equipamiento. Equipo informático con capacidad adecuada para el desarrollo de un videojuego en Unity. En cuanto al puesto de trabajo debe ser ergonómico y contará con un escritorio y una silla cómoda.

Materiales. En este tipo de recurso podríamos incluir los *assets* (recursos) que se vayan a utilizar de la tienda de *assets* de Unity (modelos 3D, texturas, animaciones...).

Espacio. El proyecto se desarrollará en mi vivienda y en la universidad. Dentro de esta categoría también se tendrá en cuenta el precio de la electricidad.

Servicios. Para el desarrollo del proyecto se usarán las siguientes herramientas externas, ya explicadas en el Capítulo 3:

- **Microsoft 365:** Requiere de una suscripción de pago. Se utilizará para redactar informes en Word, comunicación con los tutores a través de Outlook y Teams, y compartir ficheros por medio de OneDrive.
- **Astah:** Para la creación de los diagramas se hará uso de Astah Professional, siendo una versión de pago.
- **Unity:** Utilizado para el desarrollo del juego, principalmente para crear, modificar y desplegar contenido interactivo de forma eficiente.
- **Overleaf:** Utilizado para la documentación del TFG.
- **GitHub Copilot:** Esta herramienta se usará como asistencia para desarrollar código de manera más eficiente y productiva.
- **Trello:** Utilizado para la organización y planificación de las tareas del proyecto.
- **Visual Studio:** Entorno de desarrollo utilizado para escribir los *scripts* de C#, los cuales controlan el comportamiento y lógica del juego.

Tiempo. El tiempo de desarrollo tiene una fecha límite y al ser un trabajo de fin de grado no se puede aumentar la mano de obra para poder terminar antes. Por ello el tiempo que se empleará al proyecto rondará las 300 horas, como ya se ha explicado en la Sección 5.2.

Dinero. Para poder llevar a cabo el desarrollo del proyecto será necesario invertir dinero, aunque al no ser un proyecto empresarial se intentará reducir el coste al mínimo posible. Este apartado está explicado en la siguiente sección, desglosando los costes de los recursos citados anteriormente.

5.5. Plan de presupuestos

En la sección anterior se ha hablado de todos los recursos necesarios que se van a utilizar en el proyecto para que tenga éxito. A continuación, realizaremos dos planes presupuestarios, el simulado y el real. El presupuesto simulado tratará de estimar los costes que supondría este proyecto como si se tratase de un proyecto empresarial real dirigido al mercado. En el presupuesto real se tratará de estimar lo mejor posible los costes reales asociados al proyecto teniendo en cuenta su contexto académico.

5.5.1. Plan de presupuesto simulado

En primer lugar hablaremos de la **mano de obra del proyecto**. En un entorno real necesitaríamos a un desarrollador de videojuegos en Unity. El sueldo medio nacional para una

persona de estas características es de 30.499€ [36], lo que es equivalente a aproximadamente 2541€ al mes. Suponiendo que en un mes existen 22 días laborables y cada día tiene una jornada laboral de 8 horas, ganaría 14,44€ a la hora. Como en la planificación de la Sección 5.2 se ha estimado 300 horas para la realización del proyecto, el precio a pagar al desarrollador será de **4332€**. Esto corresponde al sueldo bruto, nómina que percibirá el empleado, pero la empresa además debe pagar un 33 % de ese sueldo bruto a la Seguridad Social [27] por lo que en realidad este empleado supone un coste de **5761,56€** a la empresa. La diferencia entre las dos cifras es lo que va dirigido a la Seguridad Social, **1429,56€**.

Ahora continuaremos con el equipo seleccionado para este desarrollador. Teniendo en cuenta que el ordenador debe ser adecuado para el desarrollo de videojuegos en Unity, se ha elegido un ***Lenovo Legion 5 15ITH6H*** [75]. Este portátil nuevo en la actualidad vale 1549€ y se estima una vida útil de 5 años, por lo que costaría aproximadamente 310€/año o lo que es equivalente, 25,81€/mes. Como el proyecto tiene una duración de unos 5 meses, el coste amortizado que supondrá este equipo es de **129€**.

En cuanto a los **materiales utilizados** como *assets* de la tienda de Unity dependerá de si el desarrollador quiere crear sus propios *assets*, si utilizará gratuitos o si elegirá *assets* de pago. En este caso utilizará unos *assets* que simulan una pista de atletismo que tienen un coste de **18,09€** [85]. El resto de *assets* utilizados serán gratuitos.

El **espacio de trabajo** no supondrá ningún gasto debido a que este empleado teletrabajará desde su vivienda.

En cuanto a las **herramientas externas** que debe utilizar se encuentra la licencia de **Microsoft 365 Empresa Estándar** que supone un gasto de 11,7€/mes [58] que usándolo los 5 meses serán **58,5€** en total. También se hará uso de la licencia de **Astah Professional**, en este caso es una licencia anual que vale 140€/año [7] pero como lo usaremos 5 meses supone un gasto de **58,33€**. **Unity Pro** también es anual, 1877€/año [83] lo que supone un gasto de **782€** para su uso durante toda la duración del proyecto. Por último, se adquirirá la licencia mensual de Copilot Business para el uso de **GitHub Copilot** por 19$/mes [32] siendo **87,59€** en total tras la conversión a euros. El resto de herramientas que se utilicen no son de pago o se usarán en su versión gratuita.

Concepto	Total
Sueldo bruto del empleado	4332 €
Seguridad social del empleado	1429,56 €
Portátil del empleado	129 €
Assets de la tienda de Unity	18,09 €
Licencia Microsoft 365	58,5 €
Licencia Astah	58,33 €
Licencia Unity	782 €
Licencia GitHub Copilot	87,59 €
TOTAL	**6715,07 €**

Tabla 5.15: Plan de presupuesto simulado.

Si sumamos todos los gastos anteriores obtenemos un total de **6715,07€** para el desarrollo de este proyecto dentro de un entorno empresarial enfocado al mercado. En la Tabla 5.15 se ha realizado un resumen con todos los gastos que se harían en los 5 meses que duraría el desarrollo del proyecto.

5.5.2. Plan de presupuesto real

Para el presupuesto real se seguirá la misma estructura que en el simulado, por lo que se comenzará con la **mano de obra del proyecto**. Al ser un trabajo de fin de grado la mano de obra esta formada por la alumna que lo está desarrollando, con lo cual no supone ningún gasto adicional.

El equipo que se ha seleccionado para el desarrollo de este proyecto es mi equipo habitual, un ordenador *hp laptop 15s-eq0xxx*. Este portátil nuevo valía 749€ y se estima una vida útil de 4 años, por lo que tiene un valor al año de aproximadamente 187,25€ o lo que es equivalente, 15,60€/mes. Como el proyecto tiene una duración de unos 5 meses, el coste amortizado que supondrá este equipo es de **78€**.

En cuanto a los **materiales utilizados** se utilizarán únicamente *assets* gratuitos de la tienda de Unity y el **espacio de trabajo** no supondrá ningún gasto porque se desarrollará en mi vivienda y en la universidad. No existe gasto de electricidad debido al uso de placas solares en mi vivienda.

En cuanto a las **herramientas externas** que se utilizarán como la licencia de **Microsoft 365 Empresa Estándar** o **Astah Professional** son gratuitas ya que las proporciona la propia universidad a todos los estudiantes. La licencia escogida para **Unity** es la de estudiante [84] ya que también es gratuita demostrando que eres universitario. Para el uso de **GitHub Copilot** [34] sucede exactamente lo mismo y tampoco supone un gasto adicional.

Entonces el único gasto real sería el del coste amortizado del equipo de trabajo de la estudiante siendo **78€ en total**.

5.6. Involucración de las partes interesadas

Las partes interesadas [106], también conocidas como *stakeholders*, son las personas interesadas en el proyecto. Se pueden clasificar en:

- **Internos:** Son los que pertenecen a la empresa.
- **Externos:** No pertenecen a la empresa y pueden ser clientes o usuarios que se beneficiarán del proyecto.

A continuación se identificarán a todos los *stakeholders* del proyecto junto con una breve exposición de su relación con el proyecto y que es lo que se espera que aporten cada una de ellas.

Alumna. Una de las partes más interesadas debido a que este trabajo forma parte de sus estudios de la carrera. Su papel es fundamental para el correcto desarrollo del proyecto ya que, tal y como se ha explicado en la Sección 5.1.1, tiene los roles tanto del equipo de desarrollo como de *Scrum Master*. Tomará todas las decisiones guiada por los tutores y se espera que se desarrolle el proyecto con los plazos indicados en la Sección 5.2.

Universidad de Valladolid. Institución interesada en que los alumnos contribuyan al conocimiento de sus respectivas áreas de estudio. Ya sea mediante una investigación original, análisis de casos o desarrollo de proyectos innovadores. Dentro de esta institución se encuentra el tutor académico del proyecto el cual está muy ligado al proyecto debido a su gran interés por temas relacionados con la diabetes. Se espera que colabore principalmente orientando y revisando la memoria del proyecto, participando en las reuniones de fin de *sprint* y contestando a las posibles dudas que surjan durante el transcurso de estos.

Observatorio HP-SCDS. Empresa patrocinadora del proyecto. La idea de este proyecto surge de uno de los empleados, uno de los tutores del proyecto que esta bastante ligado al él ya que sufre diabetes y al igual que el tutor académico muestra un gran interés por ello. Se espera que colabore con la alumna para orientar en la parte técnica del proyecto, también deberá participar en las reuniones de fin de *sprint* y contestar a las dudas que surjan durante el mismo.

Usuarios. Parte interesada externa al proyecto. Tienen un gran interés en él y serán los que se beneficien. Engloba tanto a pacientes de esta enfermedad como familiares, lo que lo convierte en un proyecto que ayuda a mejorar la sociedad. Se espera que lo utilicen cuando esté terminado aunque también en las fases de desarrollo se intentará que algunos usuarios interesados den *feedback* sobre el proyecto.

Profesionales de la salud. También son una parte interesada externa al proyecto. Estos profesionales se beneficiarán ya que será una herramienta muy útil que podrán utilizar para que los pacientes y familiares comprendan mejor los efectos de ingerir hidratos, aplicarse insulina o realizar ejercicio.

Administradores de los centros de salud. Otra parte interesada externa al proyecto. Tienen un gran interés por la salud pública y por todas las herramientas que puedan ayudar a los pacientes. Integrando esta herramienta en sus centros se verán beneficiados ya que la modernización de estos centros les supondrá una mejora en su *status*.

5.7. Gestión de riesgos

En el *PMBOK* se define el **riesgo** [106] como "un acontecimiento o condición inciertos que, si se producen, tienen un efecto positivo o negativo en los objetivos de un proyecto". Para reducir los efectos negativos que podrían surgir en caso de materializarse el riesgo, es fundamental realizar una buena planificación de riesgos.

Para poder distinguir los riesgos más perjudiciales y los más probables se debe llevar a cabo una evaluación de los riesgos. En nuestro caso hemos optado por utilizar la matriz de probabilidad e impacto que se encuentra en la Figura 5.4:

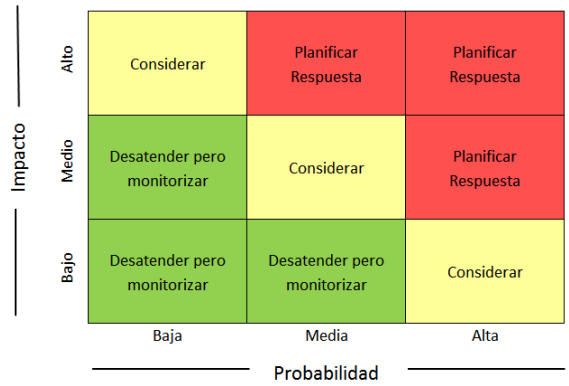

Figura 5.4: Matriz de probabilidad e impacto de los riesgos [37]

Para afrontar los diferentes riesgos, dependiendo de sus características, tenemos las siguientes opciones [106]:

- Aceptar
- Evitar
- Reducir/Mitigar
- Transferir
- Realizar acciones de contingencia

Para cada uno de los riesgos que se han identificado para este proyecto se van a tener en cuenta los siguientes campos:

- **Identificador**. Siguen el formato 'RXX' donde XX representa un número del 01 al 99. Es una forma de distinguir de manera unívoca cada riesgo.
- **Título**. Frase corta que resume el contenido del riesgo.
- **Descripción**. Párrafo más largo donde se describe en que consiste el riesgo.
- **Categoría**. 'General' o 'Particular' dependiendo de si el riesgo aplica a cualquier TFG o si es algo concreto de este.
- **Probabilidad**. Baja, Media o Alta. Describe de forma cualitativa la probabilidad de que se produzca ese riesgo.
- **Impacto**. Bajo, Medio o Alto. Describe de forma cualitativa cuánto puede afectar las consecuencias del riesgo en el proyecto si se llega a producir.
- **Riesgo**. Bajo, Medio o Alto. Según la matriz de riesgos si hay que desatenderlo, considerarlo o planificarlo.

- **Medidas de mitigación**. Conjunto de acciones que se aplicarán para reducir la probabilidad de que el riesgo se materialice.
- **Medidas de contingencia**. Conjunto de acciones que se realizarán en caso de que el riesgo ocurra para reducir al máximo el impacto de dicho riesgo.

Riesgo R01	
Título	Falta de experiencia con el lenguaje C# en Unity.
Descripción	Posibles complicaciones que surgen por la falta de conocimientos en las tecnologías en las que se desarrolla el TFG ya que son nuevas para la alumna. Lo que puede provocar retrasos o bloqueos en el desarrollo.
Categoría	Particular
Probabilidad	Media - 2
Impacto	Medio - 2
Riesgo	Medio - 2
Medidas de mitigación	▪ Formación por parte de la alumna del lenguaje C#. ▪ Realizar pequeños proyectos en Unity para familiarizarse con este motor de desarrollo de juegos.
Medidas de contingencia	▪ Preguntar al tutor experto en esas tecnologías como poder resolver la problemática que haya surgido.

Tabla 5.16: Riesgo R01. Falta de experiencia con el lenguaje C# en Unity.

Riesgo R02	
Título	Problemas al encontrar recursos gráficos.
Descripción	Dificultades para encontrar *assets* gratuitos que se adapten a la idea del proyecto provocando retrasos en el desarrollo.
Categoría	Particular
Probabilidad	Alta - 2
Impacto	Medio - 2
Riesgo	Alto - 3
Medidas de mitigación	▪ Al buscar un *asset* pensar en alternativas sencillas que también puedan adaptarse bien al proyecto.
Medidas de contingencia	▪ Buscar una de las alternativas a ese recurso. ▪ Pedir ayuda al tutor de HP-SCDS ya que él tiene acceso a más recursos.

Tabla 5.17: Riesgo R02. Problemas al encontrar recursos gráficos.

Riesgo R03	
Título	Ausencia temporal del estudiante.
Descripción	Por enfermedad o razones familiares la alumna no puede trabajar en el proyecto de manera temporal, lo que puede afectar al alcance del proyecto.
Categoría	General
Probabilidad	Media - 2
Impacto	Bajo - 1
Riesgo	Bajo - 1
Medidas de mitigación	▪ Seguir hábitos saludables. ▪ Evitar participar en actividades donde haya un alto riesgo de lesionarnos. ▪ Al hacer la planificación dejar una holgura al final del proyecto.
Medidas de contingencia	▪ Replanificar utilizando los días extra que se han dejado al final del proyecto. ▪ En caso de enfermedad leve seguir trabajando en el proyecto.

Tabla 5.18: Riesgo R03. Ausencia temporal del estudiante.

Riesgo R04	
Título	Ausencia temporal de uno o ambos tutores.
Descripción	Uno o ambos tutores no se encuentran disponibles por razones ajenas al proyecto influyendo en la calidad y alcance del mismo.
Categoría	General
Probabilidad	Alta - 3
Impacto	Medio - 2
Riesgo	Alto - 3
Medidas de mitigación	▪ Realizar reuniones periódicas para dejar planificado el trabajo durante ese *sprint*. ▪ Mantenerse comunicado con los tutores para que, si la ausencia está planificada, realizar una reunión antes. ▪ Dejar algunas tareas, que no sean prioritarias y que no se necesite ayuda de los tutores, para fases más avanzadas del proyecto.
Medidas de contingencia	▪ Avanzar con el proyecto con todas aquellas tareas que se puedan completar sin ayuda del tutor/tutores.

Tabla 5.19: Riesgo R04. Ausencia temporal de uno o ambos tutores.

Riesgo R05	
Título	Estimación optimista de las tareas.
Descripción	Realizar estimaciones optimistas del tiempo necesario para completar una tarea, lo que podría suponer retrasos o desmotivación.
Categoría	General
Probabilidad	Media - 2
Impacto	Medio - 2
Riesgo	Medio - 2
Medidas de mitigación	▪ Analizar las tareas detenidamente, identificando las partes más complejas y posibles complicaciones. ▪ Pedir opinión a los tutores.
Medidas de contingencia	▪ Monitorización y seguimiento constante del proyecto. ▪ Establecer el alcance del proyecto acorde al tiempo disponible.

Tabla 5.20: Riesgo R05. Estimación optimista de las tareas.

Riesgo R06	
Título	Problemas con el equipo de trabajo.
Descripción	El equipo de la alumna puede dejar de funcionar o quedarse sin espacio de almacenamiento impiendo así que se complete el proyecto.
Categoría	General
Probabilidad	Media - 2
Impacto	Alto - 3
Riesgo	Alto - 3
Medidas de mitigación	▪ Eliminar aplicaciones innecesarias del dispositivo para aumentar el espacio de almacenamiento disponible. ▪ Utilizar el equipo solo para el desarrollo del proyecto. ▪ Mantener todo lo relacionado con el proyecto en la nube, manteniendo el GitLab actualizado y usando Overleaf para redactar la memoria.
Medidas de contingencia	▪ Si el dispositivo se queda sin almacenamiento utilizar un disco duro externo. ▪ Si el equipo deja de funcionar, llevarlo a reparar o comprar uno nuevo.

Tabla 5.21: Riesgo R06. Fallos con el equipo de trabajo.

Riesgo R07	
Título	Problemas en la presentación del trabajo.
Descripción	En la presentación puede fallar algún programa como Unity o pasarse del tiempo permitido pudiendo penalizar en la nota del TFG.
Categoría	General
Probabilidad	Media - 2
Impacto	Alto - 3
Riesgo	Alto - 3
Medidas de mitigación	▪ Grabar la presentación para que en el caso de que algún programa falle poder reproducirlo. ▪ Practicar varias veces la presentación utilizando menos tiempo del permitido.
Medidas de contingencia	▪ Si falla algún programa reproducir el vídeo con la presentación.

Tabla 5.22: Riesgo R07. Problemas en la presentación del trabajo.

Capítulo 6

Análisis

Este capítulo engloba toda la parte de análisis del proyecto, una de las etapas fundamentales en el desarrollo de software. El análisis englobará varios temas clave del proyecto como los requisitos del proyecto, los actores involucrados junto los casos de uso, escenarios o el modelo de dominio. El análisis se ha llevado a cabo en cada una de las historias de usuario aunque la redacción de la memoria se ha hecho casi en su totalidad en el Sprint 3.

Durante este capítulo también se han incluido breves explicaciones teóricas de los elementos que han sido utilizados.

6.1. Requisitos del producto

En esta sección partimos de los requisitos proporcionados por HP-SCDS. Se realizará un análisis de esos requisitos y se adaptarán a requisitos del producto. Estos requisitos serán los que establezcan las funcionalidades y características que deben ser desarrolladas para cumplir los objetivos del proyecto. Los requisitos los podemos clasificar en requisitos funcionales y no funcionales.

6.1.1. Requisitos funcionales

Los requisitos funcionales [55] describen lo que el sistema o software debe hacer, o lo que es lo mismo, describen las funcionalidades específicas que el producto debe tener. Suelen dar respuesta a las siguientes preguntas:

- ¿Cuáles son las características que se deben implementar?
- ¿Cuáles son las tareas que debe realizar?

Comenzaremos analizando y adaptando los requisitos proporcionados por HP-SCDS (Capítulo 4) y posteriormente se definirán los requisitos del producto. Todas las modificaciones que se hagan respecto a los requisitos de HP-SCDS serán previamente comunicadas al tutor de HP-SCDS.

Adaptación de los requisitos

En el Capítulo 4 vienen explicados los requisitos de HP-SCDS pero se harán ciertos cambios que se considera que mejorarán el juego. Para no repetir en esta sección solo se explicará qué modificaciones se han hecho y en la siguiente se enumerarán todos los requisitos funcionales del producto.

- La barra de energía que representa el nivel de glucosa se ha decidido que tenga un rango de entre 0 y 300 en vez de 0 y 500. Este cambio se ha realizado ya que uno de los requisitos indica que al llegar a 300 es cuando finaliza el juego por lo que poner una barra hasta 500 podría confundir al usuario.
- La velocidad de la carrera se ha decidido que no irá aumentando en cada nivel. El objetivo es que el usuario debe aprender sobre los diferentes conceptos de la diabetes y esto es lo que irá aumentando de complejidad según se suba de nivel. Aunque la velocidad también afecta al nivel de glucosa también hará que el usuario tenga menos tiempo para pensar y eso no interesa actualmente.
- En cuanto al algoritmo de interferencia de elementos en la pista se ha tomado la decisión de que no se usará para todos los niveles. En los primeros niveles se prioriza que el usuario aprenda conceptos básicos como que si ingiere una fruta aumenta su nivel de glucosa o que si se aplica insulina bajará. Este algoritmo se usará para niveles más avanzados debido a que al coger objetos no repercute inmediatamente en el nivel de glucosa, sino que lo hace a lo largo del tiempo.
- El algoritmo de suma de puntos ha sufrido una modificación: Los alimentos que se cojan no sumarán o restarán un valor dependiendo de que tipo de alimento es, ahora sumarán o restarán dependiendo de si es una buena decisión o una mala decisión. Con decisión se entiende que si el usuario está en un rango saludable y cogiendo un elemento se mantiene en el rango saludable se considera una buena decisión por lo que se sumarán puntos. En cambio, si al coger un objeto nos salimos del rango saludable, se considerará una mala decisión y restará puntos.
- En uno de los apartados de la Subsección 4.1.3 especifica que si el jugador está por encima de 300 durante 5 segundos, terminará el maratón y se debe aplicar insulina. Esto no se ha implementado ya que podría confundir al usuario por lo que si llega a 300 se terminará el juego y aparecerá la pantalla de fin de nivel.

Identificación de los requisitos funcionales

Una vez tenemos claro que características y funcionalidades se quieren implementar para conseguir el objetivo del proyecto, estos requisitos se han plasmado en la Tabla 6.1.

ID	Título	Descripción
RF01	Menú principal	El sistema debe mostrar un menú principal donde el usuario pueda seleccionar el nivel, jugar o salir del juego. Además, dependiendo del nivel que esté seleccionado se debe mostrar información sobre dicho nivel.
RF02	Niveles	El sistemas debe tener 5 niveles cuya dificultad aumente gradualmente.
RF03	Pantalla de decisiones	El sistema debe permitir que el usuario decida el número de raciones de un tipo de hidrato que ingerirá, el número de insulinas que se aplicará y cuanto tiempo debe esperar antes de la carrera. En base a eso se calculará el nivel de glucosa inicial de cada nivel.
RF04	Carrera	El sistema debe permitir que el usuario corra por una pista a una velocidad constante donde aparezcan diferentes alimentos e insulinas y permitirle recolectarlos.
RF05	Movimiento del usuario	El sistema debe permitir al usuario moverse horizontalmente por las diferentes vías.
RF06	Cálculo nivel de glucosa	El sistema debe calcular y mostrar al usuario el nivel de glucosa actual durante la carrera.
RF07	Sistema de puntos	El sistema debe recompensar o penalizar al usuario mediante puntos la toma de decisiones o por el paso de puntos de control.
RF08	Pantalla de resumen	El sistema debe mostrar un resumen del nivel y permitir salir del juego o volver al menú principal.
RF09	Porcentaje del tiempo en rango	El sistema debe calcular el porcentaje del tiempo que el usuario está en un nivel de glucosa aceptable durante la carrera.

Tabla 6.1: Requisitos funcionales de SugarRace.

6.1.2. Requisitos no funcionales

Los requisitos no funcionales [55], también conocidos como requisitos de calidad, tienen un papel fundamental en los proyectos software. Los requisitos funcionales se centran en qué hace el software y los requisitos no funcionales en cómo lo hace. Especifican los aspectos de calidad que se deben cumplir, como la seguridad, la usabilidad, el rendimiento o la estabilidad.

También pueden abordar aspectos relacionados con la mitigación de riesgos o con la optimización de recursos, como la minimización del tiempo de respuesta o el uso eficiente de la memoria.

A continuación en la Tabla 6.2 se especifican los requisitos no funcionales de este proyecto.

ID	Título	Descripción
RNF01	Usabilidad	El sistema debe ser intuitivo y fácil de utilizar por personas con conocimientos mínimos sobre tecnología.
RNF02	Unity	El sistema debe desarrollarse en Unity.
RNF03	Compatibilidad	El sistema debe ser compatible con tecnologías de realidad virtual y con el metaverso.
RNF04	*Assets* apropiados	El sistema deberá utilizar recursos con una resolución adecuada y deben ser apropiados para el público objetivo del proyecto.
RNF05	Idioma	El sistema debe tener los textos en español.
RNF06	Estabilidad	El sistema debe funcionar continuamente sin presentar bloqueos ni fallos durante al menos 5 horas de ejecución continua en un entorno de prueba.
RNF07	Lenguaje claro y sencillo	El sistema deberá utilizar un vocabulario claro, sencillo y adecuado a las circunstancias.

Tabla 6.2: Requisitos no funcionales de SugarRace.

6.2. Actores del sistema

Un actor [73] es un usuario u otro sistema externo que puede interactuar con objetos del sistema. Los actores representan un rol que es realizado por una entidad (persona, hardware u otro sistema) que interactúa con los objetos de sus casos de uso asociados.

Al ser un juego de un solo jugador y no tener otras funcionalidades adicionales que no estén relacionadas con jugar, el sistema tendrá un único actor principal que es el usuario. El usuario es el único que va a interactuar con la interfaz y el que usará las diferentes funcionalidades proporcionadas por el sistema.

6.3. Modelo de dominio

La Figura 6.1 contiene el modelo de dominio obtenido del análisis. En él están las entidades conceptuales más significativas que se han extraído a partir de los requisitos y de los casos de usos vistos en las secciones anteriores. El modelo de dominio permite representar de una forma visual y estructurada todas las entidades, relaciones, atributos y métodos.

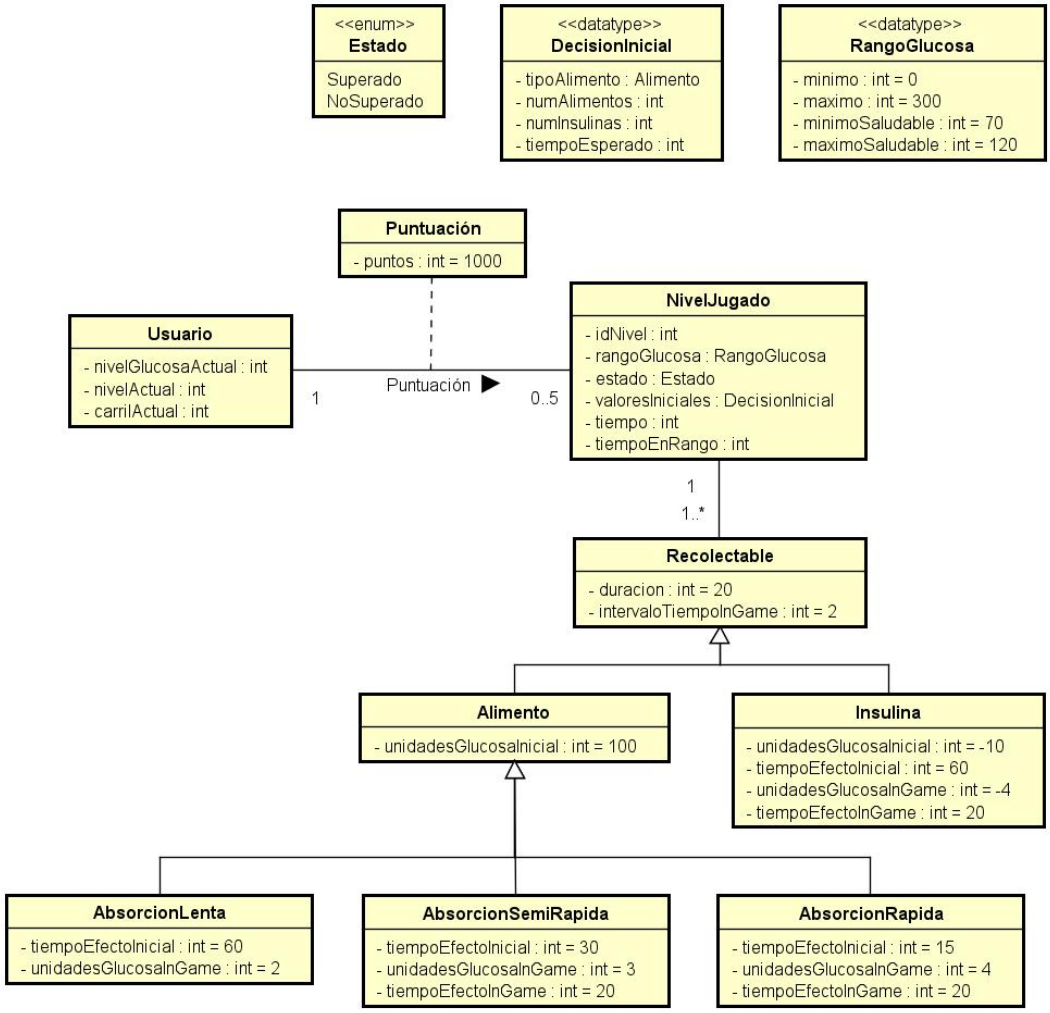

Figura 6.1: Modelo de dominio en análisis.

6.4. Casos de Uso

Los casos de uso [73] en ingeniería de software son un medio para captar los requisitos del sistema desde la perspectiva del usuario. Su función es especificar detalladamente cómo interactúan los actores con el sistema para lograr un objetivo concreto, incluyendo las acciones que realiza el usuario y las respuestas que proporciona el sistema para comprender su comportamiento esperado en diversas situaciones.

En cada caso de uso se describirá una secuencia de acciones que ocurren cuando un actor interactúa con el sistema. También se incluirá la descripción de los pasos que sigue el usuario, las opciones que están disponibles y cómo el sistema responde al usuario. Así se pretende transmitir de una forma clara y comprensible los requisitos del proyecto, lo que facilita el desarrollo del mismo.

6.4.1. Diagrama de casos de uso

El diagrama de casos de uso [42] es un tipo de diagrama de comportamiento en lenguaje UML (*Unified Modelling Languaje*) donde se pretende mostrar el comportamiento que se espera de un sistema en un caso de uso concreto. Es uno de los diagramas más estáticos ya que solo describe acciones y objetivos pero sin entrar en más detalles.

Su principal función es mostrar las relaciones entre los actores y las funciones/objetivos más importantes del sistema. Este diagrama utiliza elementos estandarizados, siendo los tres principales:

- **Actor:** Se representa con el dibujo esquemático de una persona, *stick-man*.
- **Sistema:** Se representa con un rectángulo y engloba todo lo que se refiere al sistema.
- **Casos de uso:** Se representa con una elipse que incluye una breve descripción del caso de uso.

El diagrama de casos de uso de este proyecto es el que se muestra en la Figura 6.2.

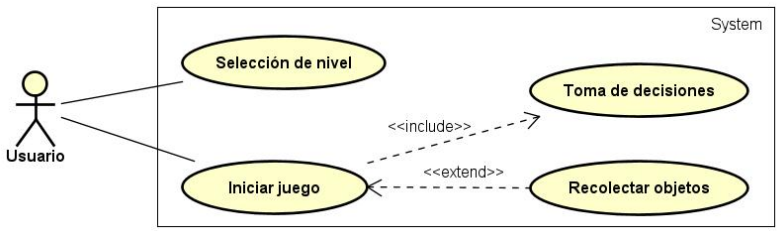

Figura 6.2: Diagrama de casos de uso.

6.4.2. Especificación de los escenarios

Un caso de uso es un conjunto de escenarios que comparten el mismo objetivo. Los casos de uso tendrán siempre un flujo normal y podrían tener flujos alternativos, a todos estos flujos se les conoce como escenarios [47]. En todos los casos de uso el actor será el usuario.

Para describir los diferentes escenarios de los casos de uso se seguirá esta estructura:

- **Caso de uso:** Breve descripción de la funcionalidad que el sistema debe realizar.
- **Descripción:** Resumen de la funcionalidad que realiza el sistema en dicho caso de uso.
- **Requisito funcional relacionado:** Enumeración de los requisitos funcionales explicados anteriormente, que están relacionados con ese caso de uso.
- **Precondición:** Condiciones o estados que se deben cumplir antes de poder ejecutar el caso de uso con éxito.
- **Secuencia normal:** Secuencia de acciones entre el actor y el sistema en una situación ideal para lograr el objetivo del caso de uso.
- **Alternativas y excepciones:** Descripción de los diferentes flujos que puede seguir el caso de uso en situaciones alternativas o excepcionales.
- **Postcondición:** Descripción de cómo debe estar el sistema después de haber ejecutado el caso de uso y se haya completado con éxito.

CU01	
Caso de uso	Selección de nivel.
Descripción	El usuario puede seleccionar un nivel en el menú principal del juego mediante flechas, lo que le permite aumentar o disminuir el nivel.
RF relacionado	RF01, RF02
Precondición	El actor debe estar en el menú principal.
Secuencia normal	1. El actor modifica el nivel. 2. El sistema aumenta/disminuye el nivel. 3. El sistema actualiza el número de nivel que aparece en la interfaz de usuario 4. Vuelve al paso 1.
Alternativas y excepciones	(1.a) El actor solicita jugar. (1.a.1) El caso de uso finaliza.
Postcondición	El nivel se ha actualizado correctamente al elegido por el usuario.

Tabla 6.3: Caso de uso 1: Selección de nivel.

CU02	
Caso de uso	Toma de decisiones.
Descripción	El usuario puede seleccionar el tipo de alimento que ingerirá y la cantidad además de la insulina que se aplicará y el tiempo que esperará para poder calcular el nivel de glucosa inicial en el juego.
RF relacionado	RF03
Precondición	El actor debe estar en la pantalla de toma de decisiones.
Secuencia normal	1. El actor solicita cambiar el tipo de alimento. 2. El sistema modifica el tipo de alimento. 3. El sistema recalcula el nivel de glucosa inicial 4. Vuelve al paso 1.
Alternativas y excepciones	(1.a) El actor solicita cambiar el número de alimentos que quiere ingerir. (1.a.1) El sistema modifica el número de alimentos. (1.a.2) Continúa en el paso 3. (1.b) El actor solicita cambiar el número de insulinas que quiere aplicarse. (1.b.1) El sistema modifica el número de insulinas. (1.b.2) Continúa en el paso 3. (1.c) El actor solicita cambiar el tiempo que quiere esperar antes de iniciar la carrera. (1.c.1) El sistema modifica el tiempo. (1.c.2) Continúa en el paso 3. (1.d) El actor solicita jugar (1.d.1) El caso de uso finaliza. (1.e) El actor solicita salir (1.e.1) El caso de uso finaliza.
Postcondición	El nivel de glucosa inicial se ha actualizado correctamente.

Tabla 6.4: Caso de uso 2: Toma de decisiones.

CU03	
Caso de uso	Iniciar carrera.
Descripción	El usuario tiene la capacidad de jugar a SugarRace. El usuario corre hacia delante automáticamente y puede controlar su movimiento horizontal y tomar la decisión de recolectar objetos. Cuando termina el juego aparecerá una pantalla de resumen del nivel.

CU03	
RF relacionado	RF04, RF05, RF06, RF07, RF08, RF09
Precondición	El actor debe estar en el menú de toma de decisiones.
Secuencia normal	1. El actor solicita iniciar la carrera. 2. El sistema carga el nivel. 3. El sistema mueve al usuario automáticamente hacia delante. 4. El sistema comprueba que está en el nivel 1. 5. El actor solicita mover el avatar a otra vía. 6. El sistema comprueba que la acción se pueda realizar. 7. El sistema mueve al avatar. 8. El actor solicita recolectar un objeto. 9. El sistema ejecuta el CU de recolectar objeto. 10. El sistema comprueba que el nivel de glucosa sea mayor que 0 y menor que 300. 11. El sistema comprueba que el tiempo sea mayor que 90 segundos. 12. El sistema muestra el resumen del nivel con el porcentaje de tiempo en rango y los puntos totales. 13. El actor solicita volver al menú principal y el caso de uso finaliza.
Alternativas y excepciones	(4.a) No es el nivel 1 (4.a.1) El sistema lanza una tarea que se ejecuta cada 2 segundos para recalcular el nivel de glucosa por la actividad física. (4.a.1) Continúa en el paso 5. (5.a) El actor no solicita moverse (5.a.1) Continúa en el paso 8. (6.a) No se puede mover. (6.a.1) Continúa en el paso 8. (8.a) El actor no solicita recolectar ningún objeto. (8.a.1) Continúa en el paso 10. (10.a) El nivel de glucosa es menor que 0 o mayor a 300. (10.a.1) Continúa en el paso 12. (11.a) El tiempo no es mayor de 90 segundos. (11.a.1) Vuelve al paso 5.
Postcondición	–

Tabla 6.5: Caso de uso 3: Iniciar carrera.

CU04	
Caso de uso	Recolectar objetos.
Descripción	El usuario recolecta un objeto que aparece en la pista.
RF relacionado	RF04
Precondición	El actor debe estar en la pantalla del juego mientras corre.
Secuencia normal	1. El actor recolecta un objeto. 2. El sistema aumenta/disminuye el nivel de glucosa. 3. El sistema modifica el nivel de glucosa de la interfaz de usuario. 4. El caso de uso finaliza.
Alternativas y excepciones	No aplica
Postcondición	El nivel se ha actualizado correctamente al elegido por el usuario.

Tabla 6.6: Caso de uso 4: Recolectar objetos.

En los casos de uso 1 y 3 cuando el actor se encuentra en alguno de los menús tendrá la posibilidad de salir del juego, al igual que en el caso de uso 2. Esto no se ha incluido en cada una de las tablas para simplificar las alternativas y así poder maquetar mejor el documento.

6.4.3. Realización en análisis de los casos de uso

A continuación se mostrarán los diagramas de secuencia que corresponden con cada uno de los casos de uso descritos anteriormente.

En la Figura 6.3 se encuentra el diagrama de secuencia del caso de uso CU01.

En la Figura 6.4 se encuentra el diagrama de secuencia correspondiente al caso de uso CU02. En este caso se ha utilizado un *loop* con la condición "!Jugar" para representar que ese *loop* se ejecuta hasta que el usuario solicite jugar (comenzar la carrera).

En la Figura 6.5 se encuentra el diagrama de secuencia correspondiente al caso de uso CU03. Aquí el usuario para poder cambiar de carril, usa la variable carril que es un entero que representa el carril al que el usuario quiere moverse.

En la Figura 6.6 se encuentra el diagrama de secuencia del caso de uso CU04.

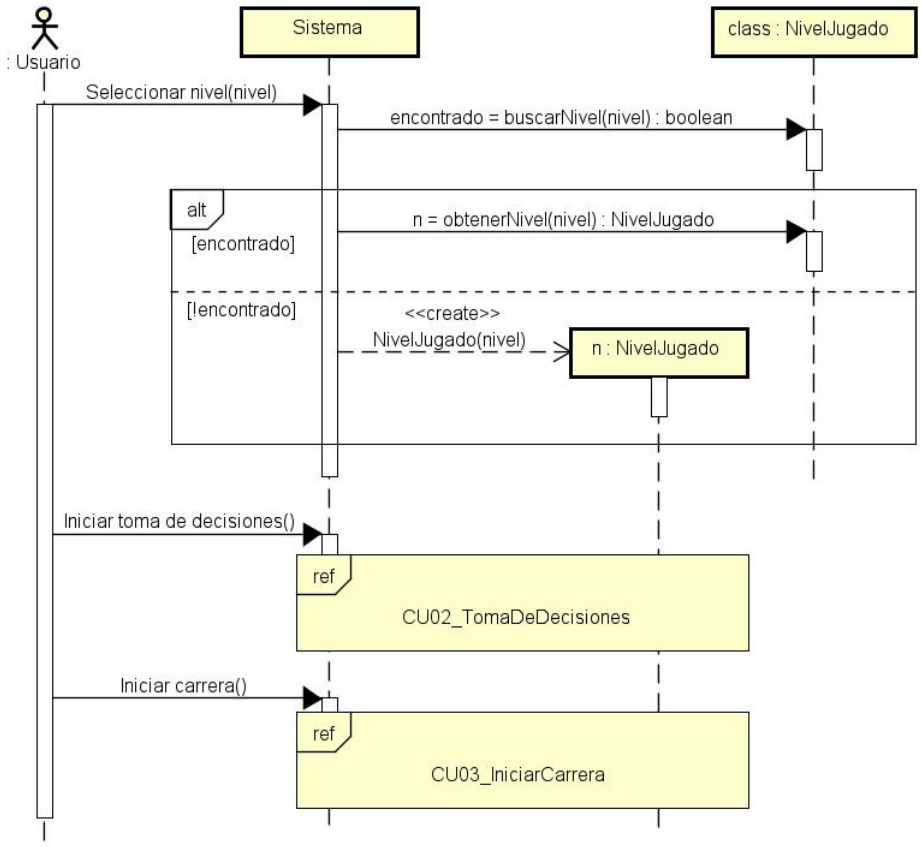

Figura 6.3: Diagrama de secuencia del caso de uso CU01, selección de nivel.

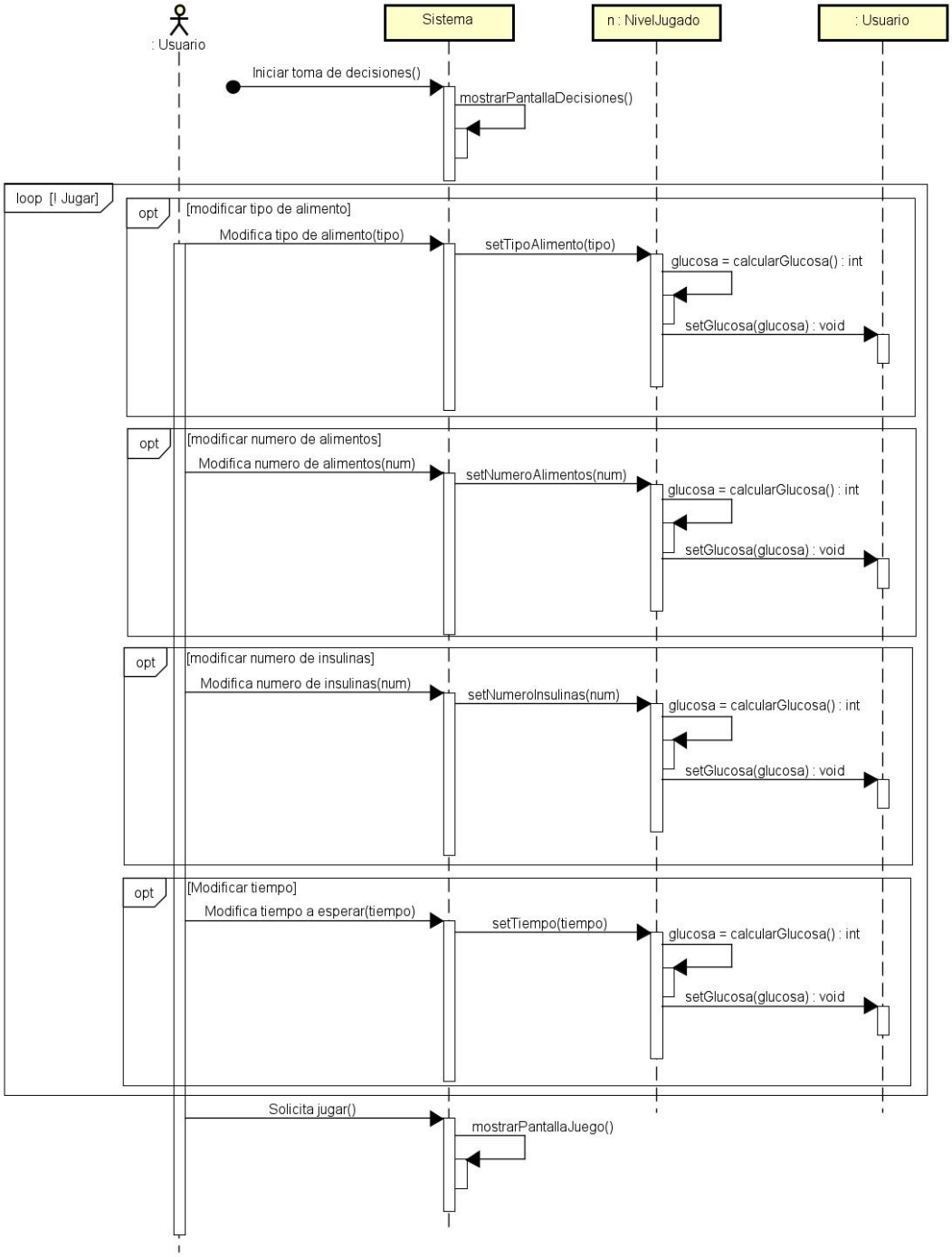

Figura 6.4: Diagrama de secuencia del caso de uso CU02, toma de decisiones.

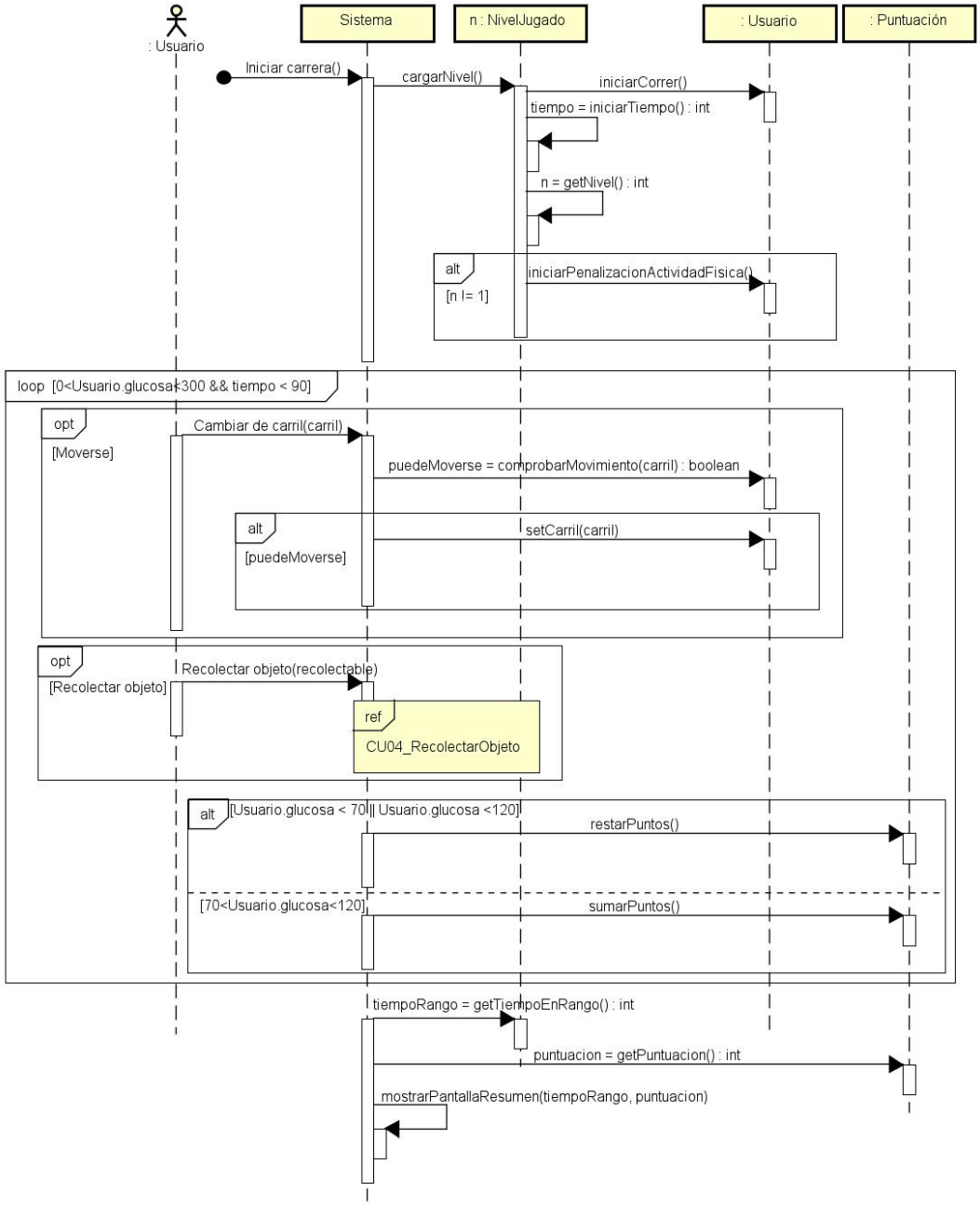

Figura 6.5: Diagrama de secuencia del caso de uso CU03, iniciar carrera.

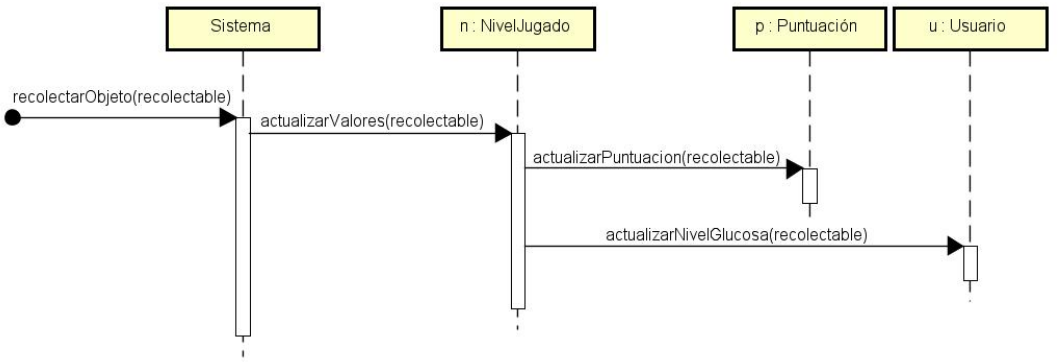

Figura 6.6: Diagrama de secuencia del caso de uso CU04, recolectar objetos.

6.5. Algoritmos utilizados

6.5.1. Algoritmo de cálculo de puntos

Este algoritmo calculará el número de puntos del usuario teniendo en cuenta tres factores, la toma de decisiones del usuario, el tiempo que está en un determinado estado y por el paso de checkpoints por la pista.

En la Figura 6.7 se ha representado con un diagrama de actividad el funcionamiento de este algoritmo.

Toma de decisiones

El usuario a lo largo del juego podrá tomar diferentes decisiones sobre si debe recolectar un alimento o una insulina. Estas decisiones deben ser tomadas en base al nivel de glucosa que tenga el avatar en ese momento por lo que ingerir un alimento o aplicarse insulina sumará o restará puntos dependiendo de la situación.

Se considerará que el usuario ha tomado una buena decisión, y que por tanto, se le sumarán 100 puntos en las siguientes situaciones:

- El avatar tiene una **hipoglucemia** y el usuario decide **ingerir un alimento**.
- El avatar está **en rango** y el usuario **ingiere un alimento o se aplica insulina** y esto hace que siga **manteniéndose en el rango** saludable.
- El avatar tiene una **hiperglucemia** y el usuario decide **aplicarse insulina**.

Sin embargo, se considerará una mala decisión y se le restarán 200 puntos si:

- El avatar tiene una **hipoglucemia** y el usuario decide **aplicarse insulina**.
- El avatar está **en rango** y el usuario **ingiere un alimento o se aplica insulina** y esto hace que **se salga del rango** saludable.
- El avatar tiene una **hiperglucemia** y el usuario decide **ingerir un alimento**.

Paso del tiempo en un estado

El usuario puede estar en tres estados diferentes mientras está en la carrera dependiendo del nivel de glucosa, tal y como se puede ver en la máquina de estados de la Figura 7.15.

Por lo tanto, el usuario será penalizado cuando pase al estado de hiperglucemia restándole 5 puntos y se irá restando 1 punto cada segundo mientras se mantenga en ese estado. Y si el usuario pasa a un estado de hipoglucemia se le restará 10 puntos y se le irán restando 2 puntos cada segundo hasta que consiga llegar al estado de normoglucemia.

En la parte superior de la Figura 6.7 se ha representado los cambios de puntuación relativos al paso del tiempo y no los relativos al cambio de estado del nivel de glucosa. Esto se ha hecho así para simplificar dicho diagrama debido a que este concepto ya se ha explicado en el párrafo anterior.

Puntos de control

El usuario mientras corre pasará por algunos puntos de control (*checkpoints*) que penalizarán al usuario si su nivel de glucosa no se encuentra en un rango saludable.

- **Glucosa en rango:** no suma ni resta puntuación.
- **Glucosa por debajo de 70:** restará 400 puntos.
- **Glucosa por encima de 120 y menor que 180:** restará 200 puntos.
- **Glucosa por encima de 180:** restará 300 puntos.

6.5.2. Algoritmo de cálculo de nivel de glucosa inicial

Antes de que comience la carrera el usuario visualizará una pantalla donde deberá tomar una serie de decisiones que influirán en el nivel de glucosa con el que empezará el juego.

En esta pantalla se podrá elegir el tipo y la cantidad de alimentos que va a ingerir, el número de insulinas que se aplicará y el tiempo simulado que esperará antes de comenzar la carrera. Todo ello será utilizado para calcular la glucosa inicial del avatar.

El nivel de glucosa que se indicará al usuario en este menú dependiendo del nivel será diferente. En los niveles 1 y 2 siempre se comenzará con 90 unidades de nivel de glucosa y en los niveles 3, 4 y 5 las unidades son aleatorias pero siempre dentro del rango 50-250 para que de esta forma el usuario se vea obligado a regular su nivel de glucosa.

Los hidratos de carbono pueden ser de tres tipos:

- **Absorción lenta:** una ración de hidratos incrementa 100 unidades el nivel de glucosa en un periodo de una hora.
- **Absorción moderada:** una ración incrementa 100 unidades en un periodo de media hora.
- **Absorción rápida:** una ración incrementa 100 unidades en un periodo de 15 minutos.

En cuanto a la insulina también existen diferentes tipos pero en el juego solo se podrá aplicar la **insulina de acción semi-rápida** que resta 10 unidades en un periodo de una hora.

El aumento y la disminución del nivel de glucosa será lineal, es decir, si tomo una ración de hidratos de carbono de absorción lenta y espero media hora, mi nivel de glucosa incrementará 50 unidades en vez de 100. Por ello el tiempo que el usuario decida esperar tendrá un papel muy importante en este cálculo.

Si el usuario espera dos horas tras haber elegido ingerir una ración de hidratos de absorción lenta, el nivel de glucosa no se verá afectado.

6.5.3. Algoritmo de recolección de objetos

Mientras el usuario juega aparecerán diferentes objetos que podrán tener un efecto sobre el nivel de glucosa del avatar. Este algoritmo será el encargado de calcularlo.

Los hidratos de carbono como ya se ha mencionado anteriormente pueden ser de tres tipos y en este algoritmo el efecto que tendrán será el siguiente:

- **Absorción rápida:** suma 4 unidades cada 2 segundos durante 20 segundos.
- **Absorción moderada:** suma 3 unidades cada 2 segundos durante 20 segundos.
- **Absorción lenta:** suma 2 unidades cada 2 segundos durante toda la partida.

La **insulina** resta 4 unidades cada 2 segundos durante 20 segundos y la **actividad física** resta 2 unidades cada 2 segundos.

El nivel finalizará después de que pase **minuto y medio** o si llega a **0 o 300 el nivel de glucosa** del avatar.

Lo especificado anteriormente será la base para los diferentes niveles donde habrá variaciones para que la complejidad del juego aumente y el usuario pueda aprender nuevos conceptos sobre la diabetes.

A continuación, se explica lo implementado en cada nivel, resaltando lo que se ha modificado de un nivel a otro.

Nivel 1

- **Solo** aparece un tipo de alimento (**Sandía**).
- Se **sumará/restará directamente** las unidades correspondientes al nivel de glucosa.
- La insulina disminuye 20 unidades al nivel de glucosa.
- La sandía aumenta 20 unidades al nivel de glucosa.
- **No** se ha implementado que reste unidades del nivel de glucosa por la **actividad física**.

Nivel 2

- **Solo** aparece un tipo de alimento (**Sandía**).
- Se **sumará/restará directamente** las unidades correspondientes al nivel de glucosa.
- La insulina disminuye 20 unidades al nivel de glucosa.
- La sandía aumenta 20 unidades al nivel de glucosa.
- **Se ha implementado que reste unidades del nivel de glucosa por la actividad física.**

Nivel 3 y 4

- **Todos los tipos de alimento pero solo hay un alimento que represente a cada tipo.**
- Se **sumará/restará las unidades al nivel de glucosa gradualmente en el tiempo.**
- Se ha implementado que reste unidades del nivel de glucosa por la **actividad física**.

Nivel 5

- **Todos los tipos de alimento pero hay varios alimentos que representan el mismo tipo.**
- Se **sumará/restará** las unidades al nivel de glucosa gradualmente en el tiempo.
- Se ha implementado que reste unidades del nivel de glucosa por la **actividad física**.

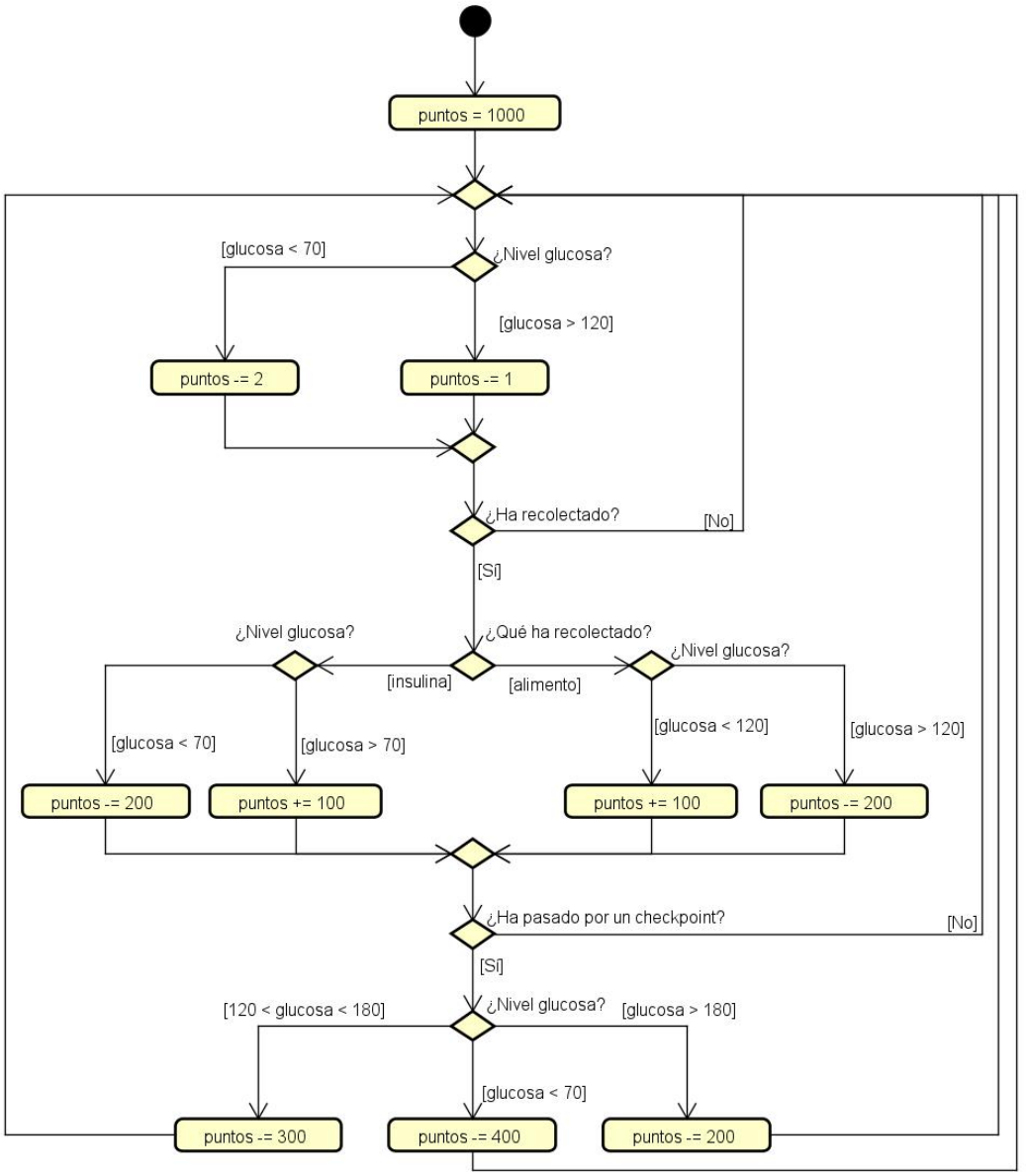

Figura 6.7: Diagrama de actividad del algoritmo de cálculo de puntos.

Capítulo 7

Diseño

En este capítulo se abordarán los temas relacionados con la parte del diseño de la solución del proyecto. A partir de la información creada en el análisis y de la toma de una serie de decisiones en base a las tecnologías utilizadas se desarrollarán los diagramas de diseño que servirán como guía para la parte de desarrollo del juego.

7.1. Arquitectura del sistema

Para este proyecto se ha decidido implementar el patrón arquitectónico ***Simple Clean Architecture*** (SCA) que sigue un estilo arquitectónico basado en capas. SCA [28] es una simplificación del patrón de diseño ***Clean Architecture*** (CA), el cual permite mantener nuestro software limpio y menos acoplado. Se ha implementado su versión simplificada debido a que si se sigue estrictamente CA en un proyecto pequeño, como es este caso, la estructura puede acabar siendo redundante.

Se ha implementado esta arquitectura para evitar dependencias excesivas con *MonoBehaviour* (explicado en el apartado 8.1) y con la manipulación en el Editor de Unity. Esto se ha hecho gracias a la separación de los componentes en capas definidas por esta arquitectura, limitar las dependencias con *MonoBehaviour* a únicamente *View* y *Presenter*, uso de la programación reactiva de forma adaptada al proyecto mediante el uso de eventos y de la inversión de dependencias para no depender de una capa interna a una externa [29].

Las relaciones entre los diferentes componentes aparecen en el diagrama de la Figura 7.1 aunque se ha tomado la decisión de no utilizar el componente *Gateway* ya que no se usará base de datos ni librerías externas actualmente. En la solución se creará el paquete de *Gateway* aunque no será utilizado. Es por ello que *UseCase* será el que tenga una dependencia con *Entity*.

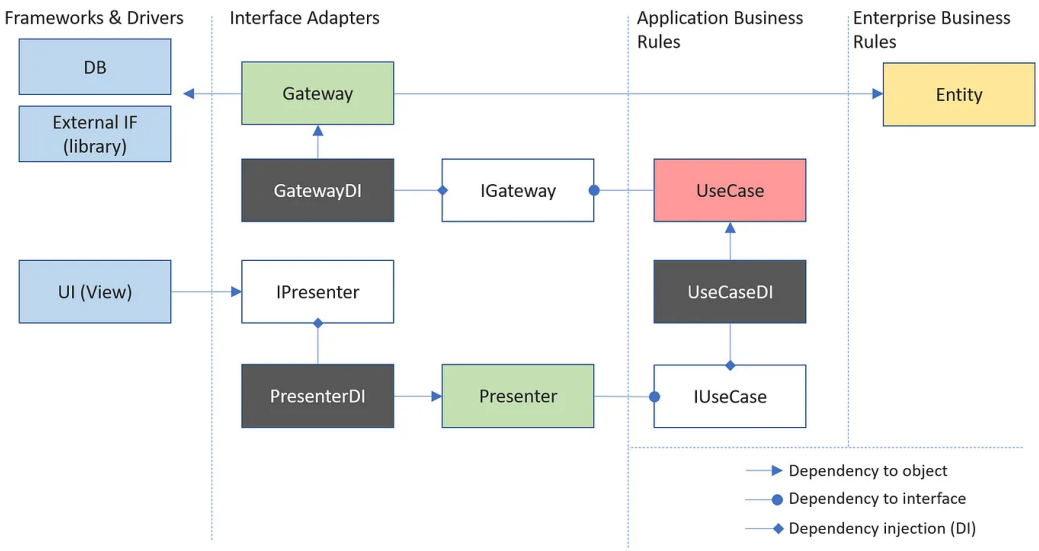

Figura 7.1: Relaciones entres componentes de la *Simple Clean Architecture* [28].

Cada uno de esos componentes tiene una responsabilidad [29]:

Entity Encapsulan las reglas de negocio más básicas. Es el componente que se encuentra en la capa más interna, por lo que no podrá tener dependencias con el resto de capas.

Use case Es la segunda capa y se encarga de modificar las entidades y de comunicarse con los *Gateway*. Solo podrá depender del *Gateway* a través de su interfaz.

Gateway Interacción con la base de datos y las bibliotecas externas. En este proyecto debido a las restricciones de tiempo no se ha utilizado.

Presenter Interacción con la Vista. Junto con el *Gateway* forman la tercera capa. Este puede hacer referencia al caso de uso para utilizar la lógica de negocio y será el encargado de actualizar la vista, gracias a la programación reactiva (explicado más adelante).

View Modifica la interfaz de usuario y es la capa más externa de todas. Únicamente podrá depender del *Presenter* a través de su interfaz.

En este tipo de arquitectura es necesaria la utilización de diferentes principios y patrones de diseño [19] para conseguir **separar las responsabilidades** (uno de los principios de SOLID) en los componentes descritos. Uno de ellos será el principio de **Inversión de dependencias**, el cual establece que los componentes de alto nivel no deben depender directamente de los componentes de bajo nivel. Para conseguir esto deberán depender de abstracciones (interfaces) y utilizar el patrón de inyección de dependencias.

También se utilizarán eventos para la comunicación entre los diferentes componentes, desacoplando así dependencias directas entre ellos y permitiendo una comunicación mucho

más flexible y escalable. Esta comunicación por eventos utiliza el enfoque de **programación reactiva** [33], un paradigma de programación basado en la propagación de cambios y manipulación de flujos de datos asíncronos.

7.2. Patrones de diseño utilizados

7.2.1. Patrón *Singleton*

El **patrón *Singleton*** [15] es un patrón de creación que permite tener **una única instancia de una clase**. Esta clase es la encargada de gestionar su instancia para asegurarse de que solo existe una. En la Figura 7.2 se muestra el diagrama correspondiente a este patrón.

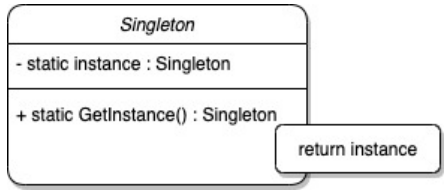

Figura 7.2: Patrón *Singleton* [15].

Este patrón se ha utilizado en las entidades, ya que al ser un juego de un único jugador y al no tener una base de datos, se ha creado una instancia del usuario que permita acceder a todos sus datos.

7.2.2. Patrón de inyección de dependencias y *Service Locator*

La **inyección de dependencias** [107] es un patrón de diseño donde una clase proporciona las dependencias de otra clase. Esto quiere decir que la responsabilidad de crear el objeto no estará en la clase que lo utiliza haciendo que el código sea más flexible.

La clase responsable de crear estos objetos es conocida como **Gestor de Inyección de Dependencias** que en esta solución se ha decidido implementar mediante el patrón *Service Locator*.

El **patrón *Service Locator*** [14] es un patrón que se utiliza para desacoplar los servicios de las clases concretas mediante la creación de un punto de acceso global a ese servicio. En la solución se crearán varias clases estáticas con propiedades públicas estáticas de las interfaces donde gracias a un constructor se inicializarán a una implementación concreta y se asignarán a dicha propiedad.

Se ha implementado en su versión más sencilla debido al tamaño y a las restricciones de tiempo del proyecto. Este patrón, en su versión completa, necesitaría una clase *Installer* que tendría la responsabilidad de registrar todos los servicios y así el *Service Locator* no tendría que estar acoplado a las implementaciones concretas.

7.2.3. Patrón observador

El **patrón Observador** [13] es un patrón de comportamiento que ayuda a desacoplar el código y cuyo principal objetivo es avisar de los cambios que se realizan en un objeto que se esté observando.

En la Figura 7.3 se muestra el diagrama de este patrón. Como el Sujeto y el Observador son abstracciones se permitirá tenerles en capas diferentes. Esto además cumple con el **principio de inversión de dependencias**, que explica que debemos depender de abstracciones y no de clases concretas.

En SugarRace se ha utilizado para notificar de que un objeto ha sido recolectado o de si se ha llegado a un punto de control en la capa *UseCase*.

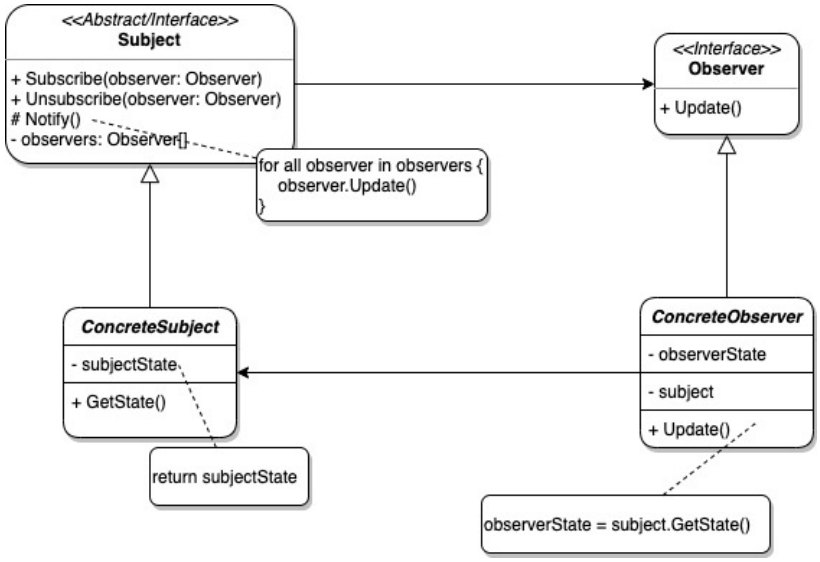

Figura 7.3: Patrón observador [13].

Además, la programación reactiva explicada en la Sección 7.1 sigue este patrón mediante el uso de eventos. Las clases que lanzan los eventos corresponden a los sujetos y las que los reciben, a los observadores. Dentro de la solución esta ha sido la forma de comunicarse entre capas sin crear dependencias directas. Normalmente se notifica un cambio que tiene que ser reflejado en la interfaz de usuario.

7.2.4. Patrón estrategia

El patrón Estrategia es un patrón de comportamiento cuyo objetivo es definir algoritmos, o clases, que puedan intercambiarse para conseguir diferentes comportamientos. De esta

forma se pretende que el consumidor no sepa que algoritmo está utilizando gracias a las abstracciones.

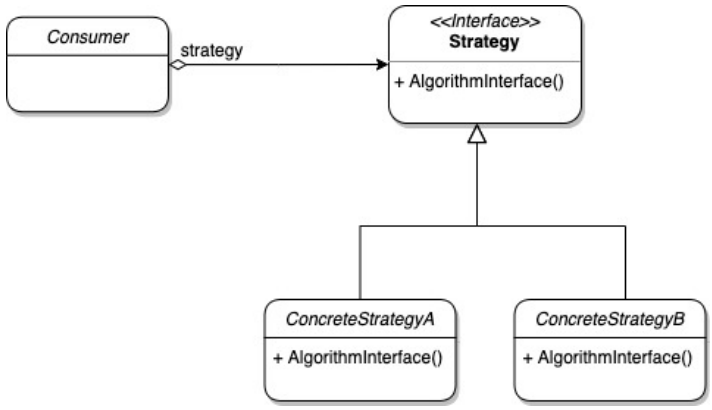

Figura 7.4: Patrón estrategia [16].

En la Figura 7.4 se encuentra el diagrama de este patrón. *Strategy* es la interfaz que implementarán todas las estrategias concretas que son las que se intercambiarán. El *Consumer* dependerá de la interfaz y mediante la inyección de dependencias no sabrá que estrategia se está utilizando. La inyección de dependencias se ha decidido hacer mediante clases auxiliares para aprender cómo funciona realmente en vez de utilizar un *framework* como *Zenject* [64] que se encarga de hacerlo de forma casi automática.

Este patrón se ha utilizado para diferenciar los algoritmos del cálculo de glucosa en los diferentes niveles del juego. De esta forma el caso de uso que se encarga de este cálculo simplemente llama a los métodos de la interfaz y no será necesario crear condiciones innecesarias que harían mucho ruido en el código.

7.2.5. Patrón *Object Pool*

Uno de los patrones de diseño utilizados para temas de optimización de recursos es el patrón *Object Pool* [12], también conocido como *Object Pooling*. Este es un patrón ampliamente utilizado en el desarrollo de videojuegos para resolver problemas de fragmentación de memoria. Esto se produce cuando se crean y se destruyen objetos ya que es una operación pesada y que requiere de memoria dinámica.

En la Figura 7.5 hay una representación de la memoria con una serie de objetos (bloques naranjas). Si creamos un Objeto A, B y A (Pasos 1, 2 y 3) y luego eliminamos el Objeto B (paso 4) quedaría un espacio vacío en la memoria, lo que produce una fragmentación de la memoria. Esto es negativo debido a que objetos más grandes no podrían ocupar dichos espacios provocando una acumulación de zonas de memoria vacías las cuales, si son muchas, pueden hacer que se pierda gran parte del espacio.

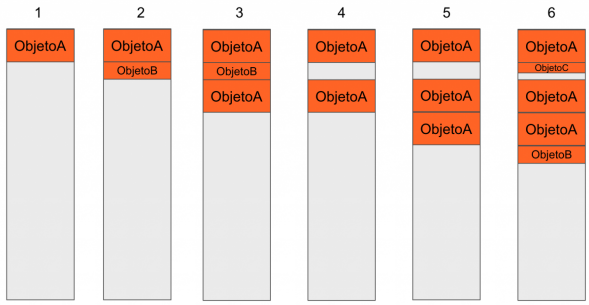

Figura 7.5: Representación de la memoria [12].

Este patrón crea los objetos una vez y no se destruyen hasta el final, lo que reduce en gran medida la fragmentación de memoria, esos pequeños espacios de memoria que se van perdiendo. Para ello se crea una colección de objetos que vamos reciclando y otra para controlar los objetos que están siendo utilizados.

En esta solución se ha utilizado para generar la pista y para generar los alimentos e insulinas que aparecen a lo largo del recorrido. Como son *GameObjects* también ha hecho falta los *prefabs* para instanciar esos objetos, estos términos se explican en la Sección 8.1.

7.3. Diagrama de paquetes

En la Figura 7.6 se encuentra el diagrama de paquetes general de toda la solución. Se incluyen también los paquetes de Unity y en especial se ha detallado el paquete Assets que es donde se ha desarrollado gran parte de la solución.

La Figura 7.7 representa el diagrama de paquetes correspondiente a la carpeta "scriptsSCA" que es donde se encuentran todos los *scripts* y clases de la solución. Durante la memoria se denominarán a todos los ficheros de dicha carpeta como *scripts* aunque los de la capa *UseCase* y *Entity* serían clases. De esta forma se simplifica la compresión de la memoria, ya que dichas clases estarán correctamente identificadas con el sufijo de la capa a la que corresponden salvo las de la capa *Entity* que no tienen ningún sufijo.

Como se puede ver existen unas capas diferenciadas que coinciden con los componentes de la SCA. Además, encontramos de color morado paquetes de terceros que han sido utilizados, *TextMeshPro* para textos y *DOTween* para las animaciones de la barra de glucosa.

También existe un paquete transversal que será utilizado por el resto de paquetes. En dicho paquete se encuentran todos aquellos ficheros que ayudan al funcionamiento del juego que deben ser accesibles desde cualquier componente. Este es el caso del fichero donde se guardan todas las constantes.

En cuanto al *Gateway* no ha sido implementado aunque si se ha dejado indicado como ya se había explicado. Por ello la relación de dependencia entre el paquete de *UseCases* y *Gateway* está en rojo.

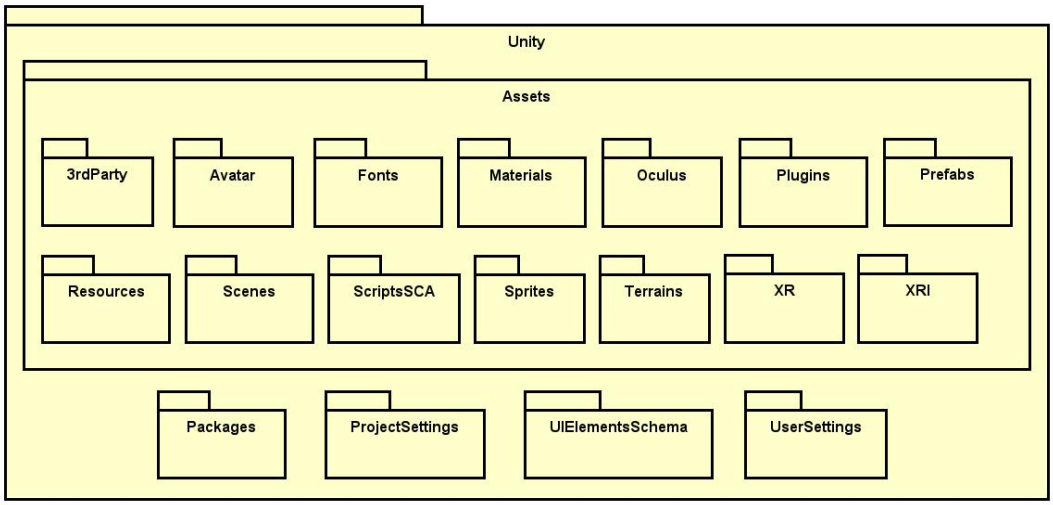

Figura 7.6: Diagrama de paquetes general de la solución.

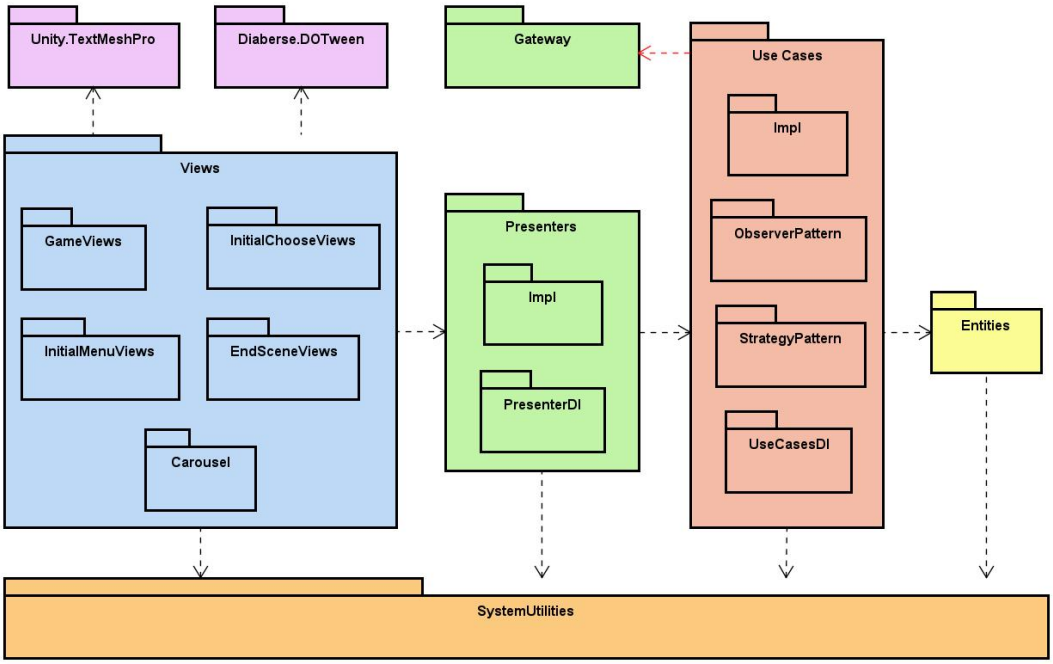

Figura 7.7: Diagrama de paquetes de la arquitectura.

En las Figuras 7.8 a 7.10 aparecen los diagramas *Decomposition&UsesStyle* de cada uno de los principales componentes a excepción de *Entities* que no tiene relaciones y *SystemUtilities* donde solo existe un fichero de constantes.

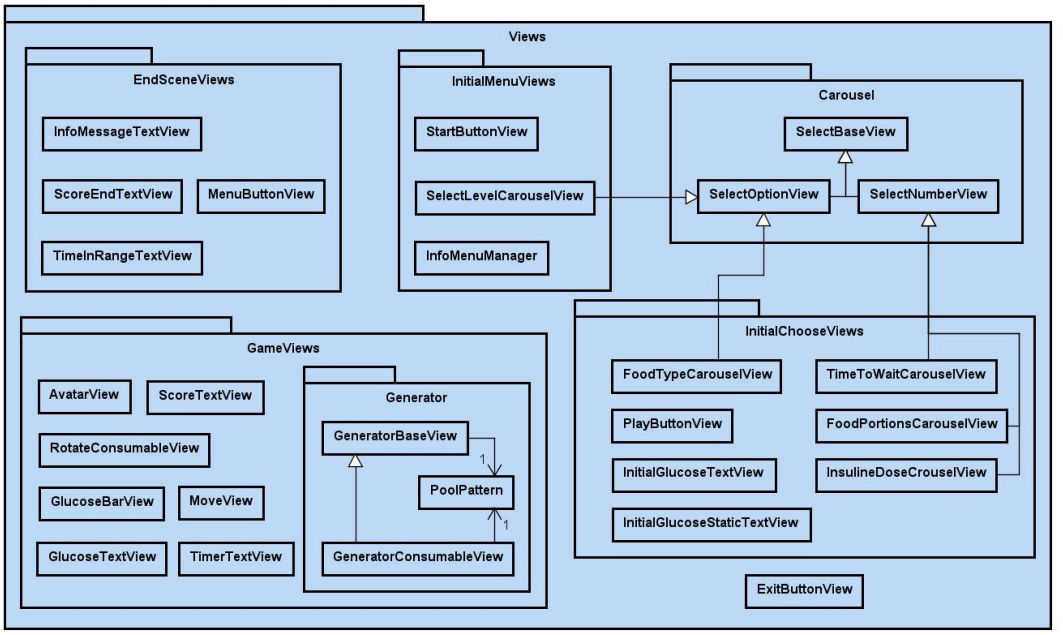

Figura 7.8: *Decomposition&Uses style* del paquete ***View***.

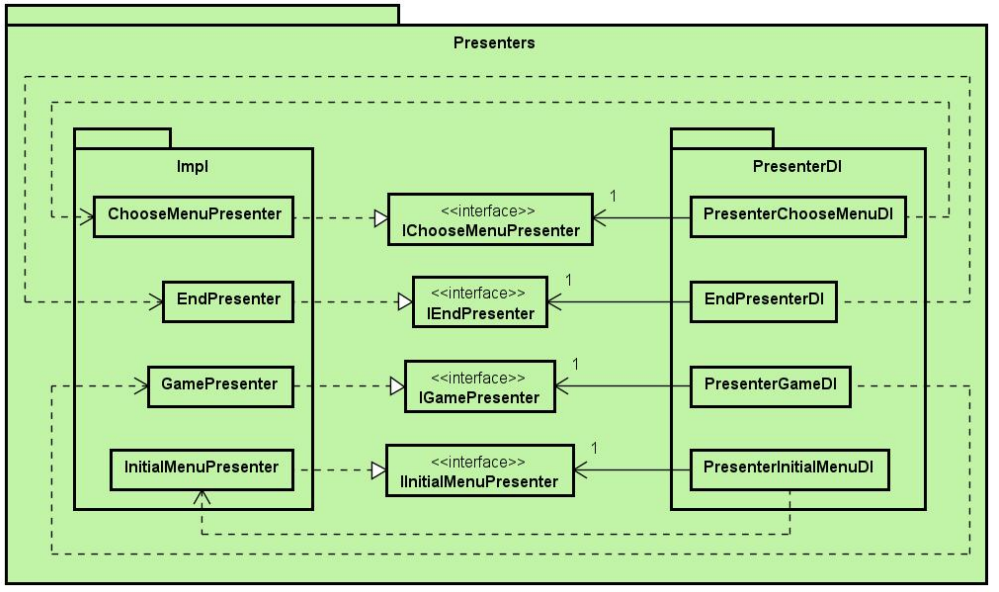

Figura 7.9: *Decomposition&Uses style* del paquete ***Presenter***.

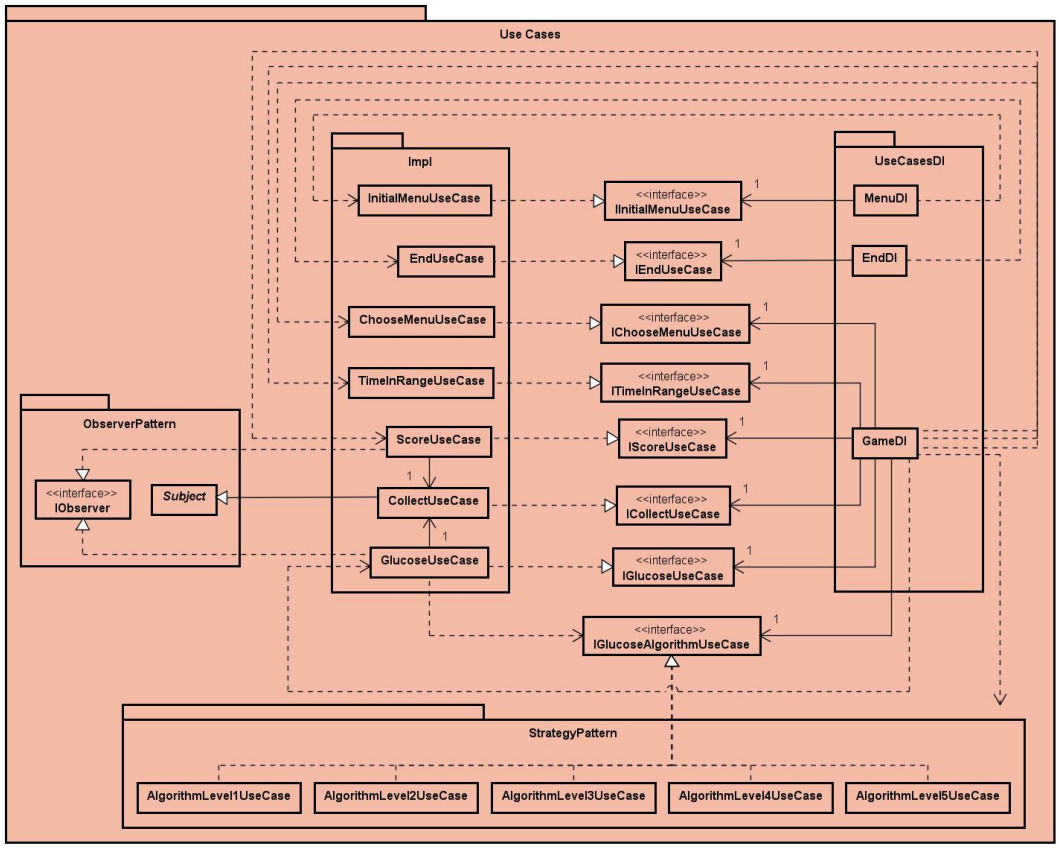

Figura 7.10: *Decomposition&Uses style* del paquete ***UseCase***.

El paquete *View*, representado en la Figura 7.8, está organizado en más paquetes que coinciden con cada una de las escenas, menos *Carousel* que es el paquete donde se encuentra los *scripts* generales de este tipo de menú y que se usa en dos escenas diferentes. El botón de salir también se encuentra en varias escenas por lo que no se encuentra en ninguno de esos paquetes.

En la Figura 7.9 y en la Figura 7.10 se puede observar como se ha implementado el patrón *Service Locator* para conseguir la inyección de dependencias. *PresenterDI* y *UseCaseDI* son los paquetes donde se encuentran las clases que implementan este patrón. En el caso del *PresenterDI* se ha decidido crear uno por cada escena del juego ya que habrá un *GameObject* en cada escena con el *script* correspondiente asignado. En cuanto a *UseCaseDI* se ha optado por unir el menú de decisiones con el juego debido a que se necesita diferenciar por nivel este menú y así se utiliza la misma instancia en todos los casos de uso.

En la Figura 7.11 se ha realizado el diagrama de *Decomposition&Uses style* del caso de uso de la pantalla del resumen del juego. En él se pueden ver todas las relaciones y dependencias existentes entre los diferentes componentes. El resto de casos de uso son muy similares por lo que por simplicidad se ha decidido únicamente mostrar este en la memoria.

Cabe destacar que no hay relaciones ni dependencias que vayan de "dentro hacía fuera", es decir, del componente *Entities* hacia *View*, lo que quiere decir que se ha respetado el principio de inversión de dependencias. Gracias a este diagrama se puede observar cómo se ha conseguido esto. Se ha creado la interfaz *IEndPresenter* que es implementada por *EndPresenter* y utilizada por las clases de la vista. De esta forma, si se quisiera modificar algún elemento de la vista no afectaría a los *Presenters* consiguiendo así desacoplar esta capa de una de más bajo nivel (*View*) gracias a que dependen de abstracciones. Sucede lo mismo entre el paquete *Presenter* y *UseCase*.

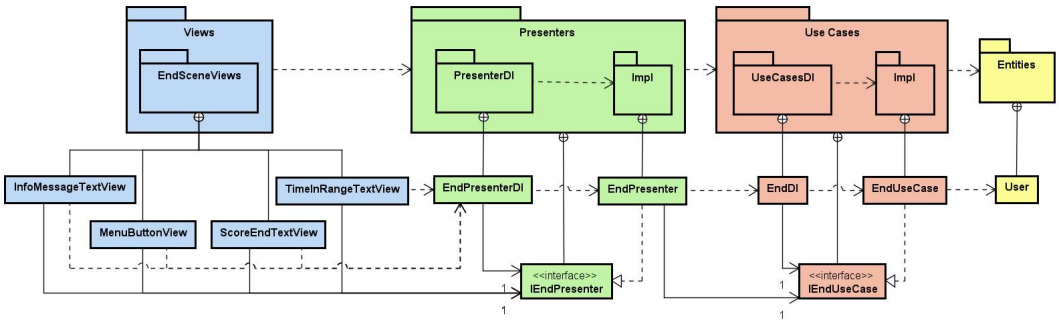

Figura 7.11: *Decomposition&Uses style* del caso de uso ***end scene***.

En los diagramas que se han mostrado anteriormente no se han representado las relaciones con Unity. En la Figura 7.12 se ha creado un pequeño diagrama de *Hierarchy style* para mostrar las relaciones y la herencia con diferentes clases del paquete de *UnityEngine*. Debido a que son muchos *scripts* del paquete *View* no se han representado todos pero todos heredan de *MonoBehaviour* y tendrán una o varias relaciones con las clases que correspondan de *UnityEngine*.

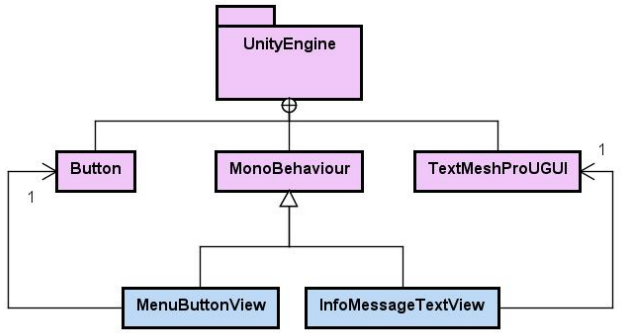

Figura 7.12: *Hierarchy style* de ejemplo de algunas clases del paquete ***View***.

En la Figura 7.13 se encuentra un diagrama similar explicado en el párrafo anterior pero en este caso sería del paquete *Presenter*. Este diagrama se ha realizado completo ya que el número de clases es mucho menor. Al ser una inyección de dependencias de la clase *Mono-*

Behaviour no se crea una instancia sino que se añade uno de los *scripts* de la implementación al *GameObject* del *PresenterDI* de la escena que corresponda.

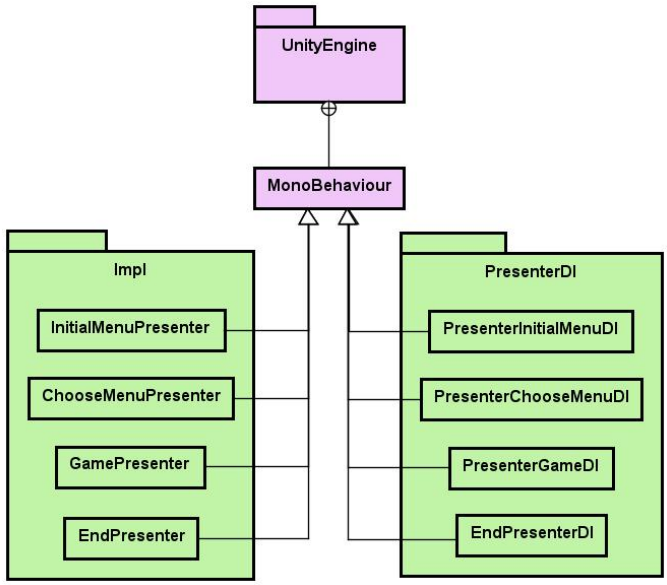

Figura 7.13: *Hierarchy style* en relación con Unity del paquete ***Presenter***.

7.4. Diagrama de componentes

Los diagramas de componentes son muy útiles para proporcionar una visión más general de nuestro sistema representando las relaciones y dependencias existentes entre los componentes. A parte también se pueden representar las interfaces proporcionadas y requeridas por cada uno de estos.

En la Figura 7.14 se está representado un diagrama de componentes del caso de uso de mostrar el resumen de la carrera. Todos los casos de uso son similares a este por ello se ha decidido incluir únicamente este diagrama en la memoria. En esa figura podemos observar que la comunicación entre las diferentes capas se realiza mediante interfaces excepto con la capa de entidades que será una dependencia.

En el resto de casos de uso también habrá varias vistas que se comunican con un presentador a través de su interfaz y que este a su vez se comunicará a través de otra interfaz con el caso de uso. Y este último será el único que conozca a las entidades. Como en el caso descrito en la Figura 7.14.

Las interfaces que han sido necesarias son las siguientes:

- **IEndPresenter:** Interfaz proporcionada por el componente *EndPrensenter* y requerida

por todos los componentes que conforman el componente *EndSceneViews*. Su objetivo es ser el punto de entrada desde la capa *View* hacía la capa *Presenter*, respetando los principios de SCA.

- **IEndUseCase:** Interfaz proporcionada por el componente *EndUseCase* y requerida por el componente *EndPresenter*. Al igual que la interfaz anterior es el punto de entrada desde la capa *Prensenter* hacía la capa *UseCase*.

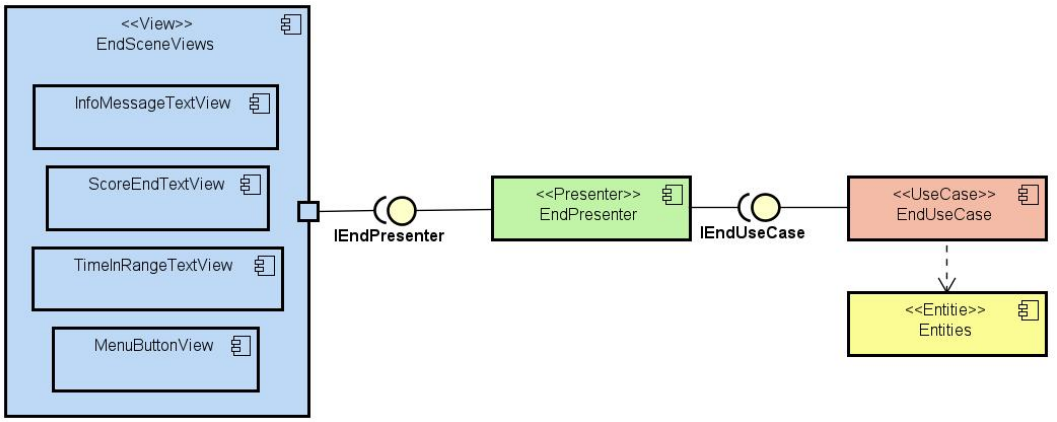

Figura 7.14: Diagrama de componentes del caso de uso del final del juego.

7.5. Máquina de estados

Analizando los requisitos se ha detectado que el usuario pasa por diferentes estados a lo largo del juego. La mejor forma de describir los cambios de estados es mediante una máquina de estados, Figura 7.15. En este caso podemos diferenciar dos partes, estados del juego o estados del nivel de glucosa.

Los estados de juego se refiere a si el usuario se encuentra en las decisiones iniciales o jugando. Los estados del nivel de glucosa se refiere a dependiendo del nivel de glucosa si está en hiperglucemia, normoglucemia o hipoglucemia. Para cambiar de un estado a otro se hará mediante el evento de recolectar ya sea alimentos o insulina. Todos los rangos que aparecen en la Figura 7.15 se refieren al nivel de glucemia.

En este diagrama faltaría indicar que por la actividad física también podría cambiar de estado pero esto no sucede en todos los niveles por ello no se ha indicado en el diagrama de la Figura 7.15. En los niveles 2-5 el nivel de glucosa del usuario bajará dos unidades cada dos segundos por realizar actividad física lo que hace posible que cambie de estado de una hiperglucemia a una normoglucemia, de una normoglucemia a una hipoglucemia y de una hipoglucemia a finalizar el nivel con el paso del tiempo.

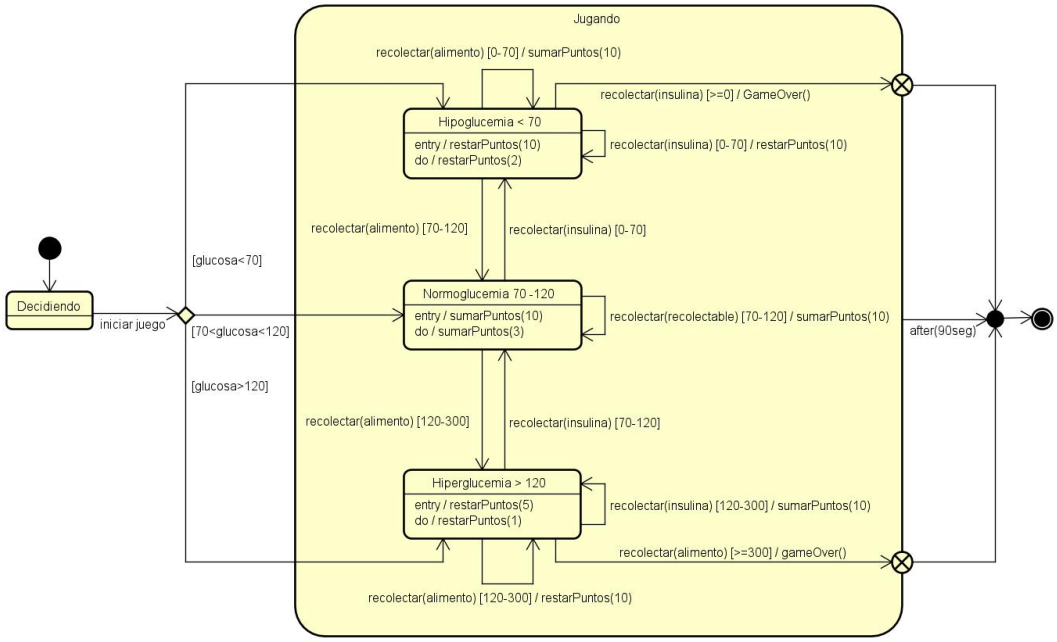

Figura 7.15: Máquina de estados.

7.6. Diagrama de despliegue

En un diagrama de despliegue [99] se muestran las configuraciones de los nodos de procesamiento en tiempo de ejecución y sus componentes. Son un tipo de diagrama de estructura que sirve para representar los aspectos físicos de un sistema.

El diagrama de despliegue de este proyecto corresponde al que se encuentra en la Figura 7.16 y en él se encuentran los siguientes elementos:

- ***VR device*** es el nodo que representa al dispositivo de realidad virtual donde se ejecutará la aplicación.
- **Unity** es el nodo que representa el entorno de ejecución de dicha plataforma y depende de los componentes de su interior.
- ***Meta XR Core SDK (UPM)*** es el artefacto que representa al conjunto de herramientas que permiten la integración de la realidad virtual en nuestro juego. Es compatible con varias plataformas de realidad virtual como *Meta Quest* o *Meta Quest 2*.
- ***XR interaction Toolkit*** es otro artefacto que representa al conjunto de herramientas proporcionadas por Unity que hace más sencilla la interacción del usuario con la realidad virtual. Proporciona funcionalidades para la interacción con objetos virtuales mediante controladores del dispositivo de realidad virtual.

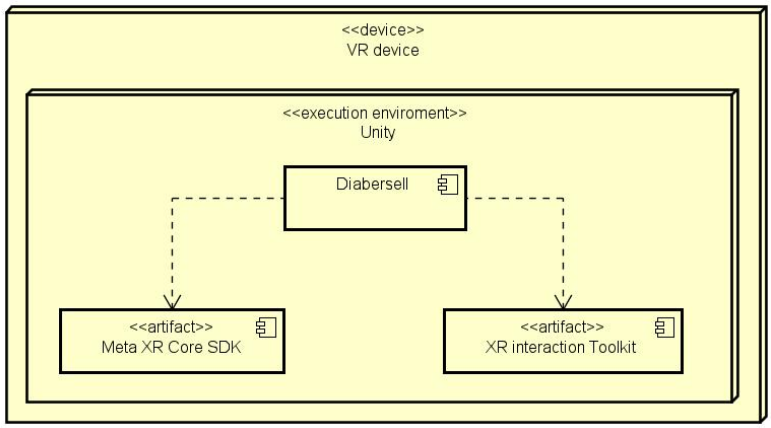

Figura 7.16: Diagrama de despliegue.

7.7. Diseño de la interfaz de usuario

Uno de los requisitos del cliente era que el juego se desarrollase en un entorno que simulará una pista de atletismo. Debido a la falta de recursos gratuitos esto no fue posible y se decidió implementar un entorno que simulará unas vías de tren junto con casas y árboles para mejorar la experiencia del usuario.

Para diseñar las diferentes interfaces de usuario se han seguido varios principios. Se ha intentando en todo momento ser **consistente** con todos los elementos de la interfaz. Se ha mantenido siempre la misma estructura en todos los niveles para facilitar a los usuarios familiarizarse con la interfaz. También se ha seguido la misma estructura entre los diferentes menús. La interfaz de usuario se ha diseñado de tal forma que sea **simple**, **fácil de entender y de utilizar**.

7.7.1. Entorno

A continuación se mostrarán las diferentes interfaces de usuario que componen el juego.

Menú inicial

En este menú el usuario podrá seleccionar el nivel que desea jugar mediante la interacción con los botones de las flechas. Tendrá la posibilidad de comenzar el nivel con el botón inferior azul o salir del juego con el botón de arriba a la derecha rojo.

En la Figura 7.17 se puede ver dicha interfaz.

Figura 7.17: Interfaz de usuario del menú inicial del juego.

Menú de toma de decisiones

El principal objetivo de esta interfaz es que el usuario elija entre diferentes opciones y que cuando haya finalizado comience la carrera. Este menú tiene una pequeña variación para los niveles 3, 4 y 5 pero se ha mantenido la consistencia de la pantalla y su usabilidad. En dichos niveles se ha modificado una de las frases y en los niveles 4 y 5 el nivel de glucosa no se recalculará por lo que es el usuario será quien deberá intentar averiguar el nivel de glucosa con el que comenzará el nivel a partir de sus propias decisiones.

En la Figura 7.18 aparece dicha interfaz con las diferentes opciones donde podrá elegir.

Figura 7.18: Interfaz de usuario del menú de decisiones del juego.

Entorno de la carrera

En esta interfaz se desarrolla la carrera del usuario. Durante el recorrido el entorno se va modificando pasando por túneles o puentes, al llegar al final de este se encuentra el punto de control indicado mediante un texto, como se puede ver en la Figura 7.19. Los laterales del mapa también varían y podrán aparecer casas, árboles o agua, si se está pasando por el puente.

Figura 7.19: Interfaz de usuario donde se muestra el punto de control durante la carrera.

Como se puede ver en la Figura 7.20 en la interfaz podemos encontrar 3 elementos muy importantes que ofrecen información al usuario en tiempo real.

- **Puntos**: Indica el número de puntos que tiene el usuario actualmente. Al inicio del nivel comienza en 1000 pero aumenta o disminuye dependiendo de las acciones que realiza el usuario mientras juega.
- **Tiempo**: En la parte superior derecha encontramos un cronómetro estilo reloj digital que indicará al usuario el tiempo que queda hasta finalizar el nivel, siempre y cuando consiga mantener su nivel de glucosa mayor que 0 y no superior a 300.
- **Nivel de glucosa**: Justo debajo del tiempo encontramos un texto junto con una barra. Estos elementos se complementan ya que la barra solo es una forma más visual de indicar el número que aparece en el texto sobre su nivel de glucosa. Esta barra aumenta o disminuye dependiendo del nivel de glucosa y será verde si este se encuentra entre 70 y 120, y rojo sino es así. El nivel de glucosa varía dependiendo de los objetos que recolecte el usuario y del ejercicio físico que realiza (dependiendo del nivel).

Mientras el usuario avanza por las vías aparecerán diferentes objetos que podrá recolectar o no. Estos tendrán diferentes repercusiones sobre su nivel de glucosa dependiendo de si toma una ración de hidratos de absorción lenta, moderada, rápida o si se aplica una insulina.

Para dar más realismo, se ha implementado un avatar corriendo gracias a la importación de una animación. Al estar en primera persona, el usuario durante el transcurso de la carrera

verá los brazos moverse por la zona inferior de la pantalla. En la Figura 7.20 se aprecia una parte de la mano izquierda de dicho avatar.

Figura 7.20: Interfaz de usuario del circuito del juego.

Menú resumen o final del juego

Este es el menú que aparece cuando el usuario termina el nivel, ya sea por terminar el tiempo o porque si nivel de glucosa haya llegado a 0 o a 300. Dependiendo de la razón se modificará el mensaje que aparece en verde en la Figura 7.21. En el caso de que no se haya superado mostrará el mensaje correspondiente y en color rojo.

Figura 7.21: Interfaz de usuario del menú de resumen del juego.

Además, aportará al usuario retroalimentación sobre el nivel jugado, mostrándole el tiempo en rango[1] y la puntuación obtenida. Por encima del 60 % este valor aparece en verde.

[1]Porcentaje de tiempo que ha estado el usuario en un rango saludable

Capítulo 8

Implementación del proyecto

En este capítulo se explicará en detalle cómo se ha llevado a cabo la implementación de cada una de las historias de usuario, explicando cada funcionalidad y desafíos o problemas encontrados pero antes de comenzar se hará una breve explicación de la clase *MonoBehaviour* y de términos de Unity utilizados en muchas de estas historias de usuario, además de explicar en términos generales como se ha seguido la *Simple Clean Architecture*.

8.1. Unity

Estos son los términos utilizados:

- **Assets**: Los *assets* [89] son cualquier elemento que se pueda utilizar en el juego. Pudiendo ser modelos 3D, imágenes, audios o cualquier tipo de archivo que se pueda utilizar en Unity.
- **GameObject**: Objetos utilizados como contenedores de componentes, que son los que implementan la funcionalidad. Todos los objetos de una escena son *GameObjects* y por sí solos no tienen ninguna funcionalidad [90].
- **Canvas**: Es un *GameObject* con un componente *Canvas* asignado y es donde deben estar todos los elementos UI [87].
- **Scene**: Contiene los menús y entornos del juego. En cada escena habrá un serie de decoraciones y entornos diferentes para poder construir el juego progresivamente [88].
- **Prefab**: Componente especial donde se guarda un *GameObject* configurado para poder reutilizarlo [93]. Se pueden utilizar en escenas diferentes y al modificar uno de ellos se podrá aplicar el cambio al resto.

MonoBehaviour

Clase base de la que deben heredar gran parte de los *scripts* de Unity [92]. Mediante el editor de Unity podemos adjuntar estos *scripts* como componentes a los *GameObjects* que nos interese. Esta clase proporciona una serie de funciones que controlan el ciclo de vida de un *GameObject*.

Aunque esta clase define muchas funciones a continuación se explicarán las que han sido utilizadas en este TFG:

- **Awake**: Se llama a este método cuando la instancia del *script* ha sido cargada.
- **Start**: Se llama antes del primer *frame* después de que *Awake* haya terminado.
- **Update**: Se llama una vez por *frame*, si el *MonoBehaviour* está activo.
- **OnDisable**: Se llama cuando se desactiva o destruye un *GameObject*. Normalmente se ha utilizado para que al terminar una escena se añadan de nuevo los valores por defecto en el modelo, parar corrutinas y dejar de escuchar eventos.

8.2. Implementación de la *Simple Clean Architecture*

En esta sección se explicará de manera general la estructura del proyecto siguiendo las directrices de la *Simple Clean Architecture* con el objetivo de no ser redundante posteriormente. Como ya se ha explicado en el Capítulo 7, esta arquitectura está dividida en capas y la comunicación entre ellas siempre es de las capas externas hacia las internas.

View

En la capa *View* todos los *scripts* heredan de *MonoBehaviour* para poder modificar la interfaz de usuario de los *GameObjects* en los que están asignados. La mayoría tiene un campo que representa el elemento visual correspondiente a ese *GameObject* y en el método *Start()* se inicializará dicho campo junto con otras acciones. Normalmente dichas acciones serán añadir el evento *OnClick* si se trata de un botón o modificar el texto del *GameObject*.

Además, todos los *scripts* donde se necesite actualizar información relativa al modelo tendrán un campo que hará referencia a la interfaz del presentador que corresponda a su escena. Este se inicializa en el método *Start()* utilizando un gestor de inyección de dependencias (explicado en la Subsección 7.2.2). **Estos gestores se encuentran en la capa *Presenter*** y existe uno por cada escena. Para identificarlos fácilmente se les ha añadido al final del nombre DI (*Dependecy Inyection*). Gracias a utilizar la inyección de dependencias conseguimos que la vista dependa únicamente de interfaces de la capa *Presenter* y por lo tanto que el código sea más flexible y se adapte mejor a los posibles cambios. En el *Awake* de los gestores se inicializa y asigna la implementación del presentador a la escena para que

al llamar al método *Start()* en los diferentes *scripts* de la capa *View* ya esté inicializado y listo para su uso (Ejemplo en el Fragmento de código 8.2).

En el caso de que haya que actualizar la vista por un cambio en el modelo se creará un evento y la vista se suscribe a él mediante el método *AddListenerXXX()* definido en el *Presenter*. De esta forma si es necesario reflejar el cambio en la interfaz, se dispara el evento y todos las vistas suscritas a él podrán realizar las acciones correspondientes.

```
[...]
public class ScoreTextView : MonoBehaviour
{
    private IGamePresenter _presenter;
    private TextMeshProUGUI _scoreText;

    private void Start()
    {
        _scoreText = GetComponent<TextMeshProUGUI>();

        _presenter = GamePresenterDI.GamePresenter;
        _presenter.AddListenerOnScoreChanged(OnScoreChanged);
    }

    private void UpdateText(int count)
    {
        _scoreText.text = "Puntos: " + count.ToString();
    }

    private void OnScoreChanged(int count)
    {
        UpdateText(count);
    }
}
```

Fragmento de código 8.1: *Script ScoreTextView*

En el Fragmento de código 8.1 se muestra un *script* que contiene los campos y métodos que se han explicado anteriormente.

Para una mejor organización se han separado en diferentes paquetes que corresponden con cada escena y, para identificarlos rápidamente, se les ha añadido al final del nombre "*View*". Además, existe la carpeta *Carousel* y el *script ExitButtonView* que son utilizados en varias escenas.

Presenter

En la capa *Presenter* también heredaran todos los *scripts* de *MonoBehaviour*. En esta capa hay una interfaz para cada escena junto con su implementación y el paquete *PresenterDI*. Será uno por cada escena en lugar de uno por funcionalidad debido a que los *scripts* de *PresenterDI* deben estar asignados a un *GameObject* invisible de dicha escena. A continuación se muestra un ejemplo en el Fragmento de código 8.2.

```
[...]
public class GamePresenterDI : MonoBehaviour
{
    public static IGamePresenter GamePresenter;

    private void Awake()
    {
        GamePresenter = gameObject.AddComponent<GamePresenter>();
    }
}
```

Fragmento de código 8.2: *Script GamePresenterDI.*

En las implementaciones existe un atributo por cada interfaz de los casos de uso que se vaya a utilizar. En el método *Awake* se inicializan estos atributos mediante los gestores de inyección de dependencias de la capa de *UseCase*, que en este caso no añade a un *GameObject* el *script* del *UseCase* sino que simplemente inicializa la clase con el constructor del *UseCase* correspondiente. La principal funcionalidad de esta capa es comunicar las entradas producidas en la vista al caso de uso y transmitir las actualizaciones realizadas en el caso de uso a la vista. En este último caso para no crear dependencias de las capas internas a las externas, se han implementado los eventos comentados anteriormente.

En esta capa se define el método *AddListenerXXX()* (Mencionado en la capa anterior) con un parámetro de entrada de tipo *Action<T>*[1]. Cuando se llama a dicho método ese parámetro de entrada se suscribe al evento *OnXXXChanged* que estará definido en el caso de uso. Si se dispara el evento se ejecutará el método delegado anónimo (el método definido en la vista). Además, el método *OnDisable()* se utiliza en todos los *Presenters* para eliminar todas las suscripciones que se hayan realizado.

En el Fragmento de código 8.3 aparece un ejemplo simplificado de lo que se ha explicado en esta capa.

[1]Encapsula un método delegado anónimo que tiene un parámetro de entrada T y no retorna ningún valor [59].

```
[...]
public class GamePresenter : MonoBehaviour, IGamePresenter
{
    public void AddListenerOnScoreChanged(Action<int> listener)
    {
        _scoreUsecase.OnScoreChanged += new EventHandler<ScoreEventArgs>
(delegate (object sender, ScoreEventArgs event_arg)
        {
            listener(event_arg.Score);
        });
    }
    [...]

    private IScoreUseCase _scoreUsecase;
    [...]

    private void Awake()
    {
        _scoreUsecase = GameDI.ScoreUseCase;
        [...]
    }

    public void UpdatePointsSeconds()
    {
        _scoreUsecase.UpdatePointsSeconds();
    }

    [...]

    private void OnDisable()
    {
        [...]

        //Unsubscribe from the event
        _scoreUsecase.OnScoreChanged = null;
        [...]
    }
}
```

Fragmento de código 8.3: Fragmentos del *script GamePresenter*.

UseCase

En la capa *UseCase* ya no se permite depender de clases de Unity consiguiendo así que la lógica, que se encuentra en esta capa, no dependa de la plataforma. Al igual que en la capa anterior tendremos interfaces, implementaciones y los gestores de inyección de dependencias (Fragmento de código 8.4).

```
[...]
public class GameDI
{
    public static IChooseMenuUseCase ChooseMenuUseCase { get; private set;
 }
    public static IScoreUseCase ScoreUseCase { get; private set; }
    public static ICollectUseCase CollectUseCase { get; private set; }
    public static IGlucoseUseCase GlucoseUseCase { get; private set; }
    public static IGlucoseAlgorithmUseCase GlucoseAlgorithmUseCase { get;
private set; }
    public static ITimeInRangeUseCase TimeInRangeUseCase { get; private
set; }

    [private static Dictionary<int, IGlucoseAlgorithmUseCase>
_algorithmMap;]

    static GameDI()
    {
        ChooseMenuUseCase = new ChooseMenuUseCase();
        CollectUseCase = new CollectUseCase();
        ScoreUseCase = new ScoreUseCase(CollectUseCase);
        GlucoseUseCase = new GlucoseUseCase(CollectUseCase);
        TimeInRangeUseCase = new TimeInRangeUseCase();

        [...]
    }
    [...]
}
```

Fragmento de código 8.4: Fragmento del *script GameDI*.

En este caso no habrá un caso de uso por escena, se ha decidido dividir por las diferentes funcionalidades (como la cuenta atrás, el cálculo del nivel de glucosa o el cálculo de los puntos). Cada una de ellas será un caso de uso que definirá los métodos necesarios para su correcto funcionamiento. En esta capa al no tener persistencia se ha decidido guardar la información directamente en las entidades explicadas en el siguiente apartado.

Para poder actualizar la interfaz cuando sea necesario se ha creado la clase *XXXEventArgs* (Fragmento de código 8.5) que hereda de *EventArgs* y que corresponde a los datos que se enviarán en el evento, facilitando así su manejo.

En la implementación de los casos de uso que necesiten usar un evento se le ha creado un campo público de tipo *EventHandler<XXXEventArgs>*. Gracias a esto el *presenter* podrá crear un manejador de eventos cuyo parámetro será de tipo *XXXEventArgs*. Cuando se

```
[...]
public class ScoreEventArgs : EventArgs
{
    public ScoreEventArgs(int score_number)
    {
        Score = score_number;
    }

    public int Score { get; set; }
}
```

Fragmento de código 8.5: Fragmento del *script IScoreUseCase*

necesite actualizar la vista simplemente se llamará al método *Invoke()* con dos parámetros, el primero el remitente del evento y luego una instancia del *XXXEventArgs*.

En el Fragmento de código 8.6 se muestra un ejemplo simplificado del código de lo que se ha explicado en el párrafo anterior.

```
[...]
public class ScoreUseCase : IScoreUseCase, IObserver
{
    public EventHandler<ScoreEventArgs> OnScoreChanged { get; set; }

    [...]

    public ScoreUseCase(ICollectUseCase collectUseCase)
    {
        [...]
    }

    public void OnNotify(string action)
    {
        UpdatePoints(action);
    }

    [...]

    private void UpdatePoints(string type)
    {
        [...]

        OnScoreChanged.Invoke(this, new ScoreEventArgs(User.GetInstance().
Score));
    }
}
```

Fragmento de código 8.6: Fragmentos del *script ScoreUseCase*

Entity

La capa *Entity* es la más estable de todas ya que no depende de ninguna otra capa. Aquí se encuentran las clases *User* y *Level* que usaran el patrón *Singleton* para tener una única instancia que este disponible en todos los casos de uso, de esta forma se guardará la información sin utilizar una base de datos.

En el Fragmento de código 8.7 se encuentra un ejemplo del código donde se pueden ver los diferentes campos y métodos y cómo se ha implementado el patrón *Singleton*.

```
[...]
public class User
{
    public static User Instance { get; private set; }

    public DiabetesState DiabetesState { get; set; }
    public int Score { get; set; }
    public double Glucose { get; set; }
    public int TimeInRange { get; set; }

    private User()
    {
        Score = Constants.InitialScore;
        TimeInRange = 0;
        Glucose = Constants.InitialGlucose;
    }

    public static User GetInstance()
    {
        if (Instance == null)
        {
            Instance = new User();
        }

        return Instance;
    }

    [...]
}
```

Fragmento de código 8.7: Fragmento del *script User*

8.3. Historia de usuario 1

Descripción: "Como desarrollador, quiero configurar el entorno de desarrollo del proyecto, para poder implementar las funcionalidades que se especifican en los requerimientos".

El objetivo de esta historia de usuario es preparar el equipo para el desarrollo del proyecto. Para ello se usarán dos entornos de desarrollo: Unity y Visual Studio.

8.3.1. Unity

Se comenzó con la instalación de Unity Hub, la versión 3.7.0 y una vez instalado, se descargó el editor de Unity en su versión estable recomendada, 2022.3.13f1, desde la ventana de *Installs* de Unity Hub, como se puede ver en la Figura 8.1. Una vez instalado el editor de Unity ya se podrá crear un proyecto, en este caso 3D, desde la ventana *Projects* del Unity Hub.

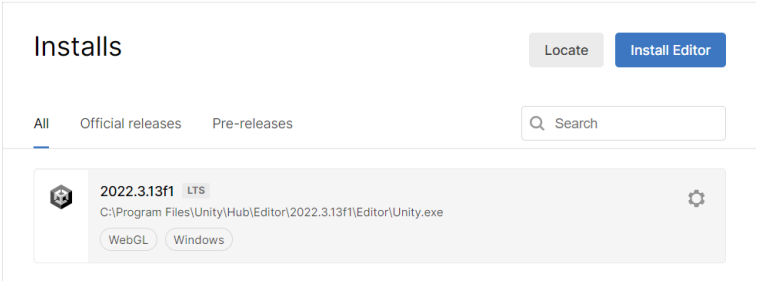

Figura 8.1: Ventana *Installs* de Unity Hub

8.3.2. *Visual Studio 2022*

Al instalar el *Unity Editor* nos da la opción de instalar el Visual Studio, en mi caso acepté y seguí los pasos hasta la pantalla de cargas de trabajo. Las cargas de trabajo son un conjunto de componentes y herramientas que se instalan juntos para soportar un tipo específico de desarrollo. En este caso se ha instalado la carga de trabajo de Desarrollo de juego con Unity, que incluye *Visual Studio Tools* para Unity, *C#* y *VisualBasic*. Como partes opcionales que también se han instalado son el centro de conectividad de Unity, *IntelliCode* y Herramientas *HLSL*.

Una vez instalado Visual Studio hay que hacer una pequeña configuración en Unity para integrarlo. En el editor de Unity seleccionamos en la parte superior izquierda la pestaña *Edit*, se abrirá un desplegable donde se debe seleccionar *Preferences*. Ahora aparecerá un menú similar al de la Figura 8.2, donde hay que seleccionar *External Tools* y en *External Script Editor* se indicará el IDE que se utilizará, en mi caso *Visual Studio 2022* [17.8.34309].

Posteriormente se descargaron las siguientes **extensiones de Visual Studio**, ya explicadas en la Subsección 3.1.3:

- *CodeMaid VS2022*.
- *GitHub Copilot*.
- *SonarLint for Visual Studio 2022*.

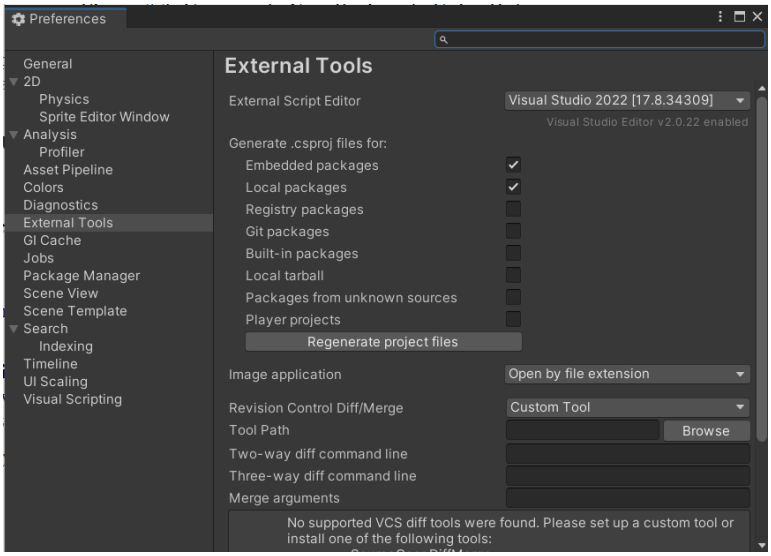

Figura 8.2: Menú *Preferences* del editor de Unity

8.4. Historia de usuario 2

Descripción: "Como desarrollador, quiero encontrar recursos para el entorno del juego, para poder crear un entorno inmersivo para el usuario".

Para esta historia de usuario se buscaron recursos existentes en la *Unity Assets Store*. En un primer momento la idea era que el entorno del juego fuera una pista de atletismo pero los únicos recursos que se adaptaban a esta idea eran de pago, por lo que esta idea quedo descartada. Aquí comenzó la búsqueda de otro entorno similar, principalmente que se adaptase a la idea de las calles de una pista de atletismo. Como ideas surgieron carreteras o vías de tren por lo que se empezó buscando esos recursos y se amplio la búsqueda a no solo la tienda oficial de Unity. Finalmente se escogió unos recursos proporcionados por la comunidad.

8.5. Historia de usuario 3

Descripción: "Como desarrollador, quiero realizar una composición de los recursos junto con el avatar, para poder simular el entorno del juego".

Una vez en la historia de usuario anterior se encontraron los recursos adecuados para el entorno se procedió a su implementación. Primero se importaron en Unity mediante el *package manager* y se creó la carpeta *3rdParty* donde se encuentran todos los recursos externos. Se desplazó la carpeta que se había creado a esta nueva carpeta.

Lo primero que se hizo fue crear mis propios *Prefabs* a partir de los importados y así

poder crear una parte de circuito más grande que es la que se irá generando. Para ello se creó un *GameObject* para cada parte del circuito como por ejemplo la carretera, el puente, el túnel o la plataforma. Se tuvo que hacer una modificación en el túnel eliminando las columnas para que así el juego se centre en la parte educativa y no en evitar obstáculos. Una vez creados estos *Prefabs* se creó otro *GameObject* que corresponde a la parte del circuito que se va generando.

Se añadió el cielo con los recursos importados de la carpeta *Sky*, para implementarlo simplemente se arrastra el material que simule el cielo a la escena. También se añadió un terreno con un material que simula el agua.

En los requisitos de HP-SCDS también se especificaba que se debían implementar puntos de control. Para que el usuario pueda identificarlos fácilmente se ha decidido añadirlos siempre después de un puente y se ha modificado uno de los *prefab* añadiendo un texto donde se indica "CHECK POINT" al finalizar el puente, como se puede ver en la Figura 8.3

Figura 8.3: Interfaz de usuario donde se muestra el punto de control durante la carrera.

8.5.1. Avatar

En esta historia de usuario también se han implementado los recursos y animaciones necesarios para el avatar que correrá durante el juego. Al igual que el resto de recursos utilizados son creados por la comunidad y gratuitos, en este caso se ha escogido un avatar de Mixamo [63], un software de Adobe que cuenta con varios modelos 3D de personajes gratuitos. Se ha seleccionado uno de ellos y se ha descargado el recurso junto con la animación de correr.

Una vez se ha descargado simplemente se puede arrastrar al editor de Unity y ya estará

listo para poder utilizarlo. Se añadirá el avatar a la escena del juego y ahora se configurara la animación para que funcione con el avatar que se ha importado. Para ello se han seguido estos pasos:

1. Seleccionar el avatar y la animación de correr y en el inspector seleccionar la pestaña *Rig*. En *Animation Type* seleccionar *Humanoid*.
2. Seleccionar la animación, en el inspector en la pestaña *Rig* de nuevo seleccionar en *Avatar Definition* la opción *Copy From Other Avatar* y en *Source* asignar el *prefab* del avatar.
3. Seleccionar la animación, en el inspector en la pestaña *Animation* seleccionar:
   a) El *checkbox Loop Time*.
   b) El *checkbox Bake Into Pose* en todas las opciones de dicha pestaña.
   c) En *Based Upon* seleccionar Original.
4. Crear una carpeta animaciones donde se debe crear un objeto de tipo *Animation Controller*.
   a) Hacer doble click y nos llevará a la pestaña *Animator*.
   b) Crear un nuevo estado seleccionando la opción *From New Blend Tree* y hacer doble click sobre él.
   c) En *Blend Type* seleccionar *2D freeform Directional*.
   d) Crear dos parámetros de tipo int denominados *SpeedX* y *SpeedY*.
   e) Añadir la animación con *Add Motion Field* y seleccionar la animación de correr con los parámetros a 0 de *SpeedX* y 0 de *SpeedY* (El personaje no se mueve, se mueve el entorno por lo que estará corriendo en el sitio).

Una vez realizadas estas configuraciones el avatar ya estará listo.

8.6. Historia de usuario 4

Descripción: "Como usuario, quiero moverme por el entorno, para poder coger o evitar objetos".

En esta ocasión se ha desarrollado un *script* denominado *Move* que usa dos funciones de Unity, *Start* y *Update*.

En el método ***Start()*** se inicializa el carril inicial y las posibles posiciones en las cuales puede encontrarse el usuario.

Durante las primeras etapas de desarrollo, cuando aún no se había implementado la realidad virtual, el método ***Update()*** utilizaba el teclado para los movimientos del personaje (en la historia de usuario 13 se explicará como se ha adaptado a la realidad virtual). En este

script no se permite que el usuario se quede entre dos carriles, tiene un movimiento discreto y únicamente se hará en el eje horizontal. Este movimiento se consigue gracias a utilizar una lista de *GameObjects*, que representan cada carril del circuito, y un índice que indica el carril actual del personaje. El índice se modifica con la entrada del usuario y tras modificarlo el método *Update()* se encarga de cambiar la posición del personaje a la de dicho carril.

8.7. Historia de usuario 5

Descripción: "Como usuario, quiero visualizar el menú principal y la pantalla de fin de nivel, para poder realizar las acciones disponibles en cada menú.".

Esta historia de usuario engloba tanto el diseño del menú principal del juego como de la pantalla de fin de nivel. A continuación se explicará cómo se hizo para cada una de ellas.

8.7.1. Menú principal

Para crear el menú principal se creó una nueva escena con un *Canvas* que contiene un panel donde se ubican los botones y las diferentes opciones. En este menú, primero se realizó la parte de interfaz de usuario para la cual se importaron *assets* de la tienda de Unity con el objetivo de mejorar el aspecto de los diferentes botones. Para dar más información al usuario se ha añadido un panel con la información relativa de cada uno de los niveles sobre los diferentes objetos que aparecen y sus efectos.

Además, se importaron recursos para implementar un menú *Carousel* que se utiliza para la selección del nivel. Utilizando como base los *scripts* importados se han creado los siguientes *scripts*:

- ***SelectBaseView*:** Clase abstracta que hereda de *MonoBehaviour* que contiene campos y métodos comunes de los menús *Carousels* junto con dos métodos abstractos implementados en las subclases para asignarlos en los botones y permitir el cambio de opción.
- ***SelectOptionView*:** Clase que hereda de la anterior y que define un *array* de *GameObjects* que serán las diferentes opciones entre las que irá pasando el menú, en este caso los 5 niveles.
- ***SelectNumberView*:** Clase que hereda también de la base y por la cual se ha necesitado crear esta jerarquía. Es similar a la anterior pero en vez de un *array* de *GameObjects* que se activan y desactivan, se asigna un *GameObject* de tipo texto al que se irá modificando el valor. Esto ha sido necesario para la historia de usuario 12 (Sección 8.14).

Para esta historia de usuario en la capa *View* se han creado los *scripts StartButtonView* y *ExitButtonView* para el manejo de los botones, *SelectLevelCarouselView* que hereda de

SelectOptionView y se encarga de llamar al *presenter* para avisar de que se ha modificado el nivel cada vez que el usuario pulsa uno de los botones y un *script* que gestiona los diferentes paneles dependiendo del nivel que haya seleccionado el usuario, gracias a que se suscribe al evento que se invoca al modificar el nivel, denominado *InfoMenuManager*.

Además, se han creado los *scripts IInitialMenuPresenter*, *InitialMenuPresenter* y *InitialMenuPresenter* de la capa *Presenter* y los *scritps IInitialMenuUseCase*, *InitialMenuUseCase* y *MenuDI* que tienen el funcionamiento ya explicado al comienzo del capítulo en la Sección 8.2.

8.7.2. Pantalla fin de nivel

Esta nueva escena sigue la estética del menú principal. Al finalizar el nivel cambia automáticamente a esta escena, en la cual hay un panel mostrando información relevante del nivel. En primer lugar aparecerá una frase que indicará al usuario si ha superado el nivel, si lo ha finalizado porque su nivel de glucosa ha llegado a 0 o a 300 o si debe reintentarlo porque su tiempo en rango (tiempo en que el usuario ha mantenido su nivel de glucosa en niveles saludables) no es superior al 60 %. Además, también se muestran los puntos finales del usuario y la frase educativa "¡Recuerda, el ejercicio puede afectar a tu nivel de glucosa hasta 24 horas después, mantente muy atento!".

Con el objetivo de que el usuario entienda mejor si se ha superado el nivel o no, el primer mensaje junto con el tiempo en rango cambiarán de color. Si se ha superado el nivel, es decir, que ha completado la carrera y ha conseguido un tiempo en rango superior al 60 %, estos aparecerán en verde y sino en rojo. Esta lógica está en la capa *UseCase*, que devuelve un booleano dependiendo de si debe ser verde o no. En el *presenter* se hace la conversión del booleano al color que corresponda de la clase de Unity y así en la vista únicamente se asigna el color al campo correspondiente.

En este caso los *scripts* creados en la capa *View* son *InfoMessageTextView*, *ScoreEndTextView* y *TimeInRangeTextView* que corresponden a los diferentes textos informativos no estáticos de la pantalla, y además, el *script MenuButtonView* que implementa la funcionalidad de ir al menú inicial cuando sea pulsado.

En la capa *Presenter* están los *scripts IEndPresenter*, *EndPresenter* y *EndPresenterDI* que tienen su análogo en la capa *UseCase*, *IEndUseCase*, *EndUseCase* y *EndDI*. En este caso una vez la pantalla se inicializa es estática por lo que no hará falta utilizar eventos, el resto del funcionamiento es similar al explicado en la Sección 8.2.

8.8. Historia de usuario 6

Descripción: "Como desarrollador, quiero encontrar recursos 3D, para poder representar alimentos e insulina".

Esta historia de usuario únicamente consistió en buscar recursos en la tienda de recursos de Unity. En cuanto a la comida fue algo más trivial ya que, aunque recursos gratuitos no hay muchos, existían varios paquetes de *assets* que contenían muchos alimentos diferentes. Se escogió un paquete de recursos 3D gratuito con 50 alimentos diferentes.

Para seleccionar el recurso que iba a simular insulina hubo más problemas. En primer lugar se acordó con los tutores evitar *assets* con aspecto de jeringuillas debido a que algunas personas pueden no estar demasiado cómodas con esta idea. Primero comenzó la búsqueda de un *asset* que simulase una pluma de insulina (Figura 8.4), pero no se encontraron *assets* gratuitos para poder seguir esta primera idea.

Figura 8.4: El elemento de la izquierda es la pluma de insulina [9] junto con la insulina.

Como segunda opción se buscaron frascos similares al que aparece en la Figura8.4 que tuvieran una pegatina que describa el contenido pero tampoco hubo éxito. Por ello se decidió buscar frascos pequeños de cualquier tipo. No fue trivial pero finalmente se encontró un *asset* que lo representaba (Figura 8.5.

Figura 8.5: *Asset* utilizado en el juego para representar la insulina.

8.9. Historia de usuario 7

Descripción: "Como usuario, quiero visualizar el entorno y los diferentes objetos en él, para poder obtener una buena experiencia de juego".

En la historia de usuario 3 se creó el entorno que simula el circuito donde corre nuestro usuario y en esta historia de usuario se ha creado el *script* que mueve el entorno, esto significa que el usuario no es el que avanza sino el entorno se mueve hacia él. Este *script* se denomina *GeneratorBaseView* y tiene dos partes importantes:

- ***OnEnable***: Es una función de Unity que es llamada justo después de que el objeto sea activado, es decir, cuando una instancia de *MonoBehaviour* se crea. En ella se realiza toda la parte de inicialización del *pool*[2] de elementos además de la creación y activación de los elementos iniciales del *pool*, calculando su posición inicial en función del índice del bucle y el desplazamiento (opcional). El número de elementos iniciales se indica mediante un parámetro llamado *Quantity* y que es configurable desde el editor de Unity.

  Todos estos elementos que se han inicializado se añaden a una cola tipo FIFO (*First In First Out*) la cual se utiliza posteriormente en el *Update*.

- ***Update***: Su objetivo es mover el generador en la dirección y velocidad especificada desde el editor de Unity mediante el atributo "*direction*" y "*speed*" además de comprobar si se necesita generar nuevos elementos, en ese caso se realizan los siguientes pasos:

  - Eliminar de la cola y desactivar el último elemento de la cola.
  - Generar un nuevo elemento en la posición adecuada mediante el uso de la posición del último elemento eliminado de la cola y moviéndolo en la dirección opuesta al movimiento del generador multiplicada por la distancia que hay entre elementos (parámetro especificado desde el editor).
  - Activación del nuevo elemento creado y añadido como primer elemento de la cola.
  - Incremento del contador del número de movimientos que se han realizado en total. Este atributo se utiliza en el *Update* para comprobar si se deben generar nuevos elementos.

Una vez creado este *script* se asigna a un *GameObject* invisible y dentro de este se crea otro que será el *pool* de objetos que se quiere que se generen. El *pool* también tiene otro *script* (*PoolPattern*) más pequeño asignado con una lista de objetos *PoolItem* editable desde el editor de Unity con los siguientes parámetros:

- El *prefab*.
- La escala del objeto (por defecto está a escala 1).
- La cantidad, número de instancias que se van a crear.Si un *PoolItem* tiene una cantidad mayor es más probable que salga a otro que tenga menor cantidad.

El *script PoolPattern* además de inicializar los objetos del *pool*, tiene un método *GetRandom* para coger un elemento aleatorio del *pool*.

[2]Patrón explicado en la Subsección 7.2.5

Para la generación de los alimentos y la insulina a lo largo del circuito se ha creado otro *script* que hereda de *GeneratorBaseView* para poder controlar dos *pool* de objetos a la vez. El funcionamiento es el mismo menos en la parte de calcular la posición donde aparecerán las insulinas que no podrá coincidir con la posición de los alimentos ya que aparecen en el mismo eje y, z pero en distinto x estrictamente para que no aparezcan objetos unos sobre otros. Otra diferencia es que el *pool* de los alimentos se asigna dinámicamente dependiendo del nivel que se haya seleccionado ya que existirá un *GameObject* por cada *pool* de alimentos necesario junto con una etiqueta que permite saber a que nivel corresponde cada uno.

8.10. Historia de usuario 8

Descripción: "Como usuario, quiero coger objetos, para poder simular la ingesta de alimentos o la aplicación de insulina".

Para llevar a cabo esta historia de usuario se realizaron los siguientes pasos:

1. A los objetos recolectables se les añadió un componente nuevo: *Box Collider*. En este componente se activó la opción de *isTrigger* que su único fin es enviar eventos cuando un cuerpo rígido entra o sale del volumen de activación [94]. De esta forma conseguimos que cuando el usuario "atraviese" uno de estos objetos, envíe un evento el cual será captado por un *script* que realizara las acciones convenientes.

2. A estos objetos también se les asignó una etiqueta identificativa para poder saber si es una insulina o el tipo de alimento. Esto permitirá identificar el objeto cuando se envíe el evento.

3. Creación del *script* que maneja el evento. Este se ha denominado *AvatarView* e irá asignado al *GameObject* del avatar. Para manejar el evento se usa el método *OnTriggerEnter* con un parámetro de entrada indicando con que *Collider* hemos chocado. Se desactiva el objeto al que pertenece ese *Collider* para que de la sensación al usuario de que se ha cogido y aquí surge una de las primeras decisiones de diseño. Seguramente cuando cojamos un alimento o una insulina se querrá modificar el nivel de glucosa o los puntos en base a ello pero meter toda esta lógica en este *script* no sería correcto ya que los *scripts* deben tener una única funcionalidad por el principio de responsabilidad única. Para solucionar este problema sin crear dependencias innecesarias entre *scripts*, se ha implementado el patrón observador a nivel de la capa de *UseCase*.

    - ***Subject***: Clase abstracta que contiene una lista de observadores y 3 métodos generales: *AddObserver*, *RemoveObserver* y *NotifyObservers*. El *script CollectUseCase* hereda de esta clase y cuando llega el evento *OnTriggerEnter* desde la vista, pasa por el *presenter* y llega al método *OnCollect()* en el caso de uso donde se usa el método *NotifyObservers()* con un parámetro para indicar que objeto se ha recolectado.
    - ***IObserver***: En este caso esta clase es una interfaz con un único método: *OnNotify()*. Las clases que implementen esta interfaz serán las encargadas de decidir la funcionalidad que tendrá. En este caso será el *script GlucoseUseCase* que se

explicará en la historia de usuario 9 y *ScoreUseCase* que se explicará en la historia de usuario 10.

Además, para que se vea más atractivo se creó otro *script*, *RotateConsumableView*, que se asignará a todos los objetos que se recolectan para que roten sobre sí mismos. Este *script* es muy sencillo y simplemente tiene un atributo velocidad modificable desde el editor de Unity y en el método *Update()* se usará uno de los métodos que tiene el *transform* denominado *Rotate* y que sirve precisamente para eso.

8.11. Historia de usuario 9

Descripción: "Como usuario, quiero saber mi nivel de glucosa, para poder mantenerlo en un nivel saludable".

Esta historia de usuario se divide en tres partes, diseño de la UI de la barra de glucosa, implementación de animaciones y la implementación de la funcionalidad.

8.11.1. Diseño de interfaz de usuario

La primera idea fue buscar en la tienda de Unity *assets* recursos similares a barras de energía o de vida pero no se encontró nada que se adaptara adecuadamente a la idea que se tenía. Aquí comenzó una pequeña investigación de cómo hacer una barra propia sin utilizar recursos de terceros. Aunque la primera impresión fue que iba a ser más laborioso se descubrió que muchos desarrolladores de Unity utilizaban un simple *sprite* de un cuadrado para crear una barra sencilla (Figura 8.6) siguiendo estos pasos:

1. Creación del *Canvas*que contenga todos los elementos de UI.
2. Creación de un *GameObject* vacío denominado *GlucoseBar*, que dentro de él están los siguientes *GameObjects* de tipo UI ->*Image*:
   - ***GlucoseBarFill***: parte de la barra que se mueve gracias al *script GlucoseBarView* y la que tiene los colores más vivos. El rango de nivel de glucosa saludable, entre 70 y 120, tiene un color verde y entre 0-70 y 120-300 será rojo.
   - ***Backgrounds***; Para que la barra de glucosa sea más intuitiva se ha puesto un fondo detrás de la barra de glucosa anterior debido a que para simular el nivel de glucosa esa barra se hace más pequeña horizontalmente y sin tener un fondo se vería el entorno. De esta forma cuando se haga más pequeña se verá el color de la barra principal pero mucho más apagado y así el usuario también sabe donde está el rango correcto.
3. Dentro del *Canvas* también hay *GameObject* de UI de tipo *text* encima de la barra de glucosa que indica con un valor cuál es el nivel de glucosa exacto. Este valor numérico es modificado mediante el *script* de *GlucoseTextView*.

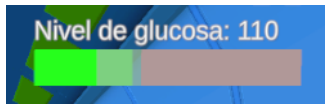

Figura 8.6: Barra de glucosa creada junto con su valor numérico en la parte superior.

8.11.2. Implementación de la funcionalidad

Por un lado, en la capa *View* existen dos *scripts* mencionados anteriormente, *GlucoseBarView* y *GlucoseTextView*, cuya funcionalidad es mostrar el nivel de glucosa del usuario. En el caso del texto simplemente es cambiar el valor y en el caso de la barra sería modificar el atributo *fillAmount* al inicializarse y durante el juego mediante un método de *DOTween* (explicado en el siguiente apartado). Estos dos *scripts* se suscriben al evento *OnCurrentGlucoseChanged*.

En la capa *Presenter* al igual que en el resto de historias de usuario se ha creado la interfaz *IGamePresenter*, la implementación *GamePresenter* y para la inyección de dependencias *GamePresenterDI*. En este caso hay una pequeña variación respecto a lo que se explicó en la Sección 8.2 para solucionar un problema que surgió al añadir los diferentes niveles en el juego. Los tres últimos niveles suman o restan al nivel de glucosa cada X segundos una pequeña cantidad en vez de sumar todas las unidades de golpe al coger cualquier objeto. Esto se ha implementado añadiendo corrutinas a nivel de *Presenter* cuando el evento se dispara. El problema entonces es que al tener dos *scripts* suscritos a dicho evento, las llamadas se duplicaban por lo que se puso un *boolean* para que solo una de ellas se encargase de activar esas corrutinas cuando fuera necesario. La implementación de este método se muestra en el Fragmento de código 8.8.

Este método también tiene la responsabilidad de comprobar que el nivel de glucosa esté entre 0 y 300 (*CheckLevelGlucose* en el Fragmento de código 8.8) y, en caso contrario, cambiar a la escena de fin de nivel. El parámetro *active* es el *boolean* que se ha explicado en el párrafo anterior y de si se trata de un alimento de absorción lenta habrá que llamar al método *InvokeRepeating()* cuyo fin es llamar al método que se indique como parámetro junto con cuantos segundos debe esperar para empezar y cada cuantos segundos debe repetirse. En este caso empezará directamente (0 segundos) y se llamará cada 2 segundos. Finalizará cuando se llame al método *OnDisable()*. Si se trata de cualquier otro alimento o de insulina entonces se activa una corrutina que es similar al método anterior pero termina después de 20 segundos. Si hubiera corrutinas activas al terminar el nivel también terminan en el método *OnDisable()*.

Por otro lado, en la capa *UseCase* están los *scripts* *IGlucoseUseCase*, *GlucoseUseCase* y *GameDI* de esta historia de usuario. En este caso el funcionamiento es el ya explicado en la Sección 8.2 pero añadiendo que *GlucoseUseCase* implementa también *IObserver*. Por ello implementa el método *OnNotify()* donde recibe como parámetro el tipo de objeto recolectado y modifica el nivel de glucosa.

```
[...]
public void AddListenerOnCurrentGlucoseChanged(Action<int> listener, bool
active)
{
    _glucoseUseCase.OnCurrentGlucoseChanged += new EventHandler<
CurrentGlucoseEventArgs>(delegate (object sender, CurrentGlucoseEventArgs
event_arg)
    {
        CheckLevelGlucose(event_arg.CurrentGlucose, Constants.MinGlucose,
Constants.MaxGlucose);
        if (!active && event_arg.Type != null)
        {
            if (event_arg.Type == Constants.SlowFood)
            {
                InvokeRepeating(nameof(
UpdateGlucoseIncrementalyCollectSlow), 0f, 2f);
            }
            else
            {
                StartCoroutine(InvokeGetIncrementGlucoseValue(event_arg.
Type));
            }
        }
        listener(event_arg.CurrentGlucose);
    });
}
[...]
```

Fragmento de código 8.8: Fragmento del *script GamePresenter*

8.11.3. Implementación de la animación

Para que cuando se actualice la barra de glucosa no sea tan brusco visualmente se ha añadido una animación mediante *DOTween* que hace que disminuya o aumente lentamente a una velocidad que se puede modificar desde el editor de Unity. También el cambio de color se hará más suavemente. Para ello se han usado los métodos *DOFillAmount* y *DOColor* de este *plugin* externo.

8.12. Historia de usuario 10

Descripción: "Como usuario, quiero ganar o perder puntos, para tener una experiencia de juego más divertida".

Para esta historia de usuario se ha creado en la capa *View* el *script*, *ScoreTextView* asignado al *GameObject* de tipo texto incluido en el *Canvas* de la escena del juego. Su responsabilidad es mostrar la puntuación del usuario mientras juega.

También se han utilizado los *scripts IGamePresenter* y *GamePresenter* donde se ha añadido los métodos y eventos necesarios para el funcionamiento de esta nueva funcionalidad, el

GamePresenterDI que no ha hecho falta ninguna modificación y los *scripts* de la capa de *UseCase* que son *IScoreUseCase*, *ScoreUseCase* y *GameDI*. En el ejemplo de la implementación de la *Simple Clean Architecture* en la Sección 8.2 se explica como se ha implementado.

El *script ScoreUseCase* sería el otro *script* que implementa la interfaz *IObserver* por lo que tiene el mismo funcionamiento que *GlucoseUseCase* explicado en la sección anterior.

8.13. Historia de usuario 11

Descripción: "Como desarrollador, quiero crear un algoritmo diferente para cada nivel sobre el cálculo del nivel de glucosa, para poder enseñar diferentes conceptos sobre la diabetes en cada nivel.".

Esta historia de usuario no contiene ninguna parte visual, es decir, aunque haya 5 niveles no existe una escena para cada nivel por lo que la interfaz de usuario será la que se creó en la historia de usuario de la Sección 8.5.

Debido a que en cada nivel existen pequeñas variaciones se ha implementado el patrón estrategia explicado en la Subsección 7.2.4. Para llevar a cabo su implementación se han creado 5 *scripts*, uno por cada nivel, denominados *AlgorithmLevelXUseCase* donde la X corresponde al nivel. Estos implementan la interfaz *IGlucoseAlgorithmUseCase* la cual define un contrato de lo que deben implementar los *scripts* de todos los niveles.

Para definir el algoritmo adecuado dependiendo del nivel se realiza mediante inyección de dependencias utilizando el *script GameDI*. En el constructor del mismo se ha creado un diccionario con clave el número de nivel y valor una instancia de la implementación que corresponda, de este modo será más sencillo asignar la implementación correcta. Para modificar el nivel si el usuario decide jugar, se ha creado un método a parte, que también se llama en el constructor de *GameDI*, dependiendo del nivel seleccionado utiliza el diccionario para asignar la implementación de dicho nivel, tal y como se muestra en el Fragmento de código 8.9.

Este patrón se utiliza en el menú de las decisiones iniciales explicado en la sección 8.14 para saber si se debe actualizar el nivel de glucosa en tiempo real o no y si el valor inicial del nivel de glucosa es aleatorio o comienza en 90. También se utiliza durante el juego para:

- Saber si debe restar unidades de glucosa el ejercicio físico.
- Saber si los objetos tienen un efecto inmediato en el nivel de glucosa o si, al contrario, se suma/resta de manera incremental.
- Indicar el pool de objetos correspondiente a cada nivel.
- Indica cuántas unidades de glucosa se debe sumar por aplicarse insulina o recolectar cualquier tipo de alimento.

```
[...]
public class GameDI
{
    public static IChooseMenuUseCase ChooseMenuUsecase { get; private set;
}
    public static IGlucoseUseCase GlucoseUseCase { get; private set; }
    public static IGlucoseAlgorithmUseCase GlucoseAlgorithmUseCase { get;
private set; }
    [...]

    private static Dictionary<int, IGlucoseAlgorithmUseCase> _algorithmMap;

    static GameDI()
    {
        [...]
        _algorithmMap = new Dictionary<int, IGlucoseAlgorithmUseCase>
        {
            { 1, new AlgorithmLevel1UseCase() },
            { 2, new AlgorithmLevel2UseCase() },
            { 3, new AlgorithmLevel3UseCase() },
            { 4, new AlgorithmLevel4UseCase() },
            { 5, new AlgorithmLevel5UseCase() }
        };

        SetGlucoseAlgorithm();
    }

    public static void SetGlucoseAlgorithm()
    {
        if (_algorithmMap.ContainsKey(GetLevel()))
        {
            GlucoseAlgorithmUseCase = _algorithmMap[GetLevel()];
            GlucoseUseCase.SetGlucoseAlgorithm(GlucoseAlgorithmUseCase);
            ChooseMenuUsecase.SetGlucoseAlgorithm(GlucoseAlgorithmUseCase);
        }
    }

    private static int GetLevel()
    {
        return GlucoseUseCase.GetLevel();
    }
}
```

Fragmento de código 8.9: Fragmento del *script GameDI*

En el Fragmento de código 8.10 aparece los métodos definidos en la interfaz *IGlucoseAlgorithmUseCase*.

De esta forma cuando se necesite utilizar en los *scripts GamePresenter* y *ChooseMenuPresenter* ya estará inicializado y dichos *scripts* no sabrán que implementación están utilizando reduciendo así dependencias. En el método *Awake()* de dichos *scripts* será necesario llamar al método *SetGlucoseAlgorithm()* para que cada vez que se cargue dicha escena actualice la implementación al nivel seleccionado por el usuario.

```
[...]
public interface IGlucoseAlgorithmUseCase
{
    int GetIncrementGlucoseValue(string type);

    bool ShouldPenalizeExercise();

    bool ShouldAddGlucoseInstantly();

    bool IsHiddenGlucose();

    string GetPoolName();

    int GetInitialGlucoseValue();
}
```

Fragmento de código 8.10: Fragmento del *script* *IGlucoseAlgorithmUseCase*

8.14. Historia de usuario 12

Descripción: "Como usuario, quiero decidir cuántas raciones de hidratos ingerir, cuánto tiempo esperar y cuántas unidades de insulina aplicar, para poder calcular mi nivel de glucosa inicial del juego.".

Esta historia de usuario ha requerido de la creación de una nueva escena entre el menú inicial y el juego en sí. Dicha interfaz tendrá un aspecto similar al del resto de los menús y contiene un *Canvas* con un panel donde se han añadido **4 menús *Carousel*** diferentes permitiendo así al usuario elegir el **tipo de alimento** y el **número de raciones** que quiere ingerir, el **número de insulinas** que quiere aplicarse y finalmente el **tiempo en minutos** que va a esperar antes de comenzar a realizar el ejercicio físico. Esto implica que existe un *script* por cada *Carousel*. El elemento que muestra el tipo de alimento hereda de *SelectOptionView* y el resto de *SelectNumberView*, en este último caso tanto los alimentos y las insulinas se suman de uno en uno al dar al botón y el del tiempo irá de 15 en 15 minutos para que suponga un cambio más apreciable. Estos *scripts* corresponden con *FoodPortionCarouselView*, *FoodTypeCarouselView*, *InsulineDoseCarouselView* y *TimeToWaitCarouselView*.

Además, existe un *GameObject* de tipo texto que indica al usuario su nivel de glucosa inicial, aunque dependiendo del nivel, se actualizará mientras el usuario modifica los menús *Carousel* o no, añadiendo así más dificultad en los últimos niveles. Esto corresponde con los *scripts* *InitialGlucoseStaticTextView* e *InitialGlucoseTextView*. Para poder jugar se ha añadido un botón que al ser pulsado cambia de escena, *PlayButtonView*.

En la capa *Presenter* se han creado los *scripts* *IChooseMenuPresenter*, *ChooseMenuPresenter* y *ChooseMenuPresenterDI*. En cuanto a la capa *UseCase* se han creado *IChooseMenuUseCase* y *ChooseMenuUseCase*. Para la inyección de dependencias se ha utilizado *GameDI* para no duplicar el código de la instanciación del patrón estrategia explicado en la Sección 8.13. El funcionamiento de todos estos *scripts* es igual al ya explicado en la Sección 8.2. La única diferencia existente es en el número de parámetros del método delegado

que se llama al recibir el evento (Fragmento de código 8.11).

```
[...]
public void AddListenerOnInitialGlucoseChanged(Action<int, Color> listener
)
{
    _usecase.OnInitialGlucoseChanged += new EventHandler<
InitialGlucoseEventArgs>(delegate (object sender, InitialGlucoseEventArgs
event_arg)
    {
        listener(event_arg.Glucose, GetTextColor(event_arg.Glucose));
    });
}
[...]
```

Fragmento de código 8.11: Fragmento del *script ChooseMenuPresenter*

8.15. Historia de usuario 13

Descripción: "Como usuario, quiero sumergirme en una experiencia orientada al metaverso y en realidad virtual durante el juego, para poder aprender sobre la diabetes de una forma interactiva y realista.".

Durante esta historia de usuario se llevó a cabo la implementación de la realidad virtual en la aplicación. Esto ha implicado instalar diferentes extensiones y configurar y adaptar ciertas partes de la aplicación.

A continuación se detallan los pasos realizados para implementar la realidad virtual en este proyecto:

1. Instalación de XR Plugin Management, el cual se encarga de configurar, gestionar y optimizar la integración de extensiones de realidad extendida[3] en este tipo de proyectos en Unity. Se establece *OpenXR* y *Mock HMD Loader* como los *Plugin Providers.*

2. Instalación de XR Interaction Toolkit, herramientas para la interacción del usuario con el juego en realidad virtual.

3. Agregar el componente de interacción *XR Simple Interactable* a los *GameObjets* con los que el usuario puede interactuar (botones). De esta forma se permite que los objetos respondan a eventos de interacción de los controladores de realidad virtual.

4. Agregar en todas las escena un *GameObject* de tipo *XR Origin* para controlar y establecer el punto de origen donde se coloca la cámara o el usuario en la escena. Una

[3]Este término engloba a todas las tecnologías inmersivas, realidad virtual, realidad aumentada y realidad mixta.

vez añadida hay que establecer el *Tracking Origin Mode* donde establecemos que referencia espacial se va a utilizar. En este proyecto se estableció el suelo como el nivel de referencia para la posición y la orientación del usuario.

Al crear este *GameObject* aparecen dentro de él una cámara, el "*Left Controller*" y el "*Right Controller*". Estos últimos representan a los dispositivos de entrada que pueden utilizar los usuarios para interactuar con la realidad virtual. Gracias a estos el usuario podrá pulsar los botones o moverse durante el juego, como se puede ver en la Figura 8.7.

5. Añadir el componente *Input Action Manager* con el paquete *XRI Default Input Actions* al *XR Origin* para poder gestionar las entradas del usuario asignando acciones a los controladores de realidad virtual. De esta forma, la configuración de las interacciones y de los controles se facilita en gran medida.

6. Al añadir *XR origin* a la escena se crea otro automáticamente denominado *XR Interaction Manager* con un componente asignado del mismo nombre. Su principal funcionalidad es detectar los eventos relacionados con los controladores de realidad virtual.

7. Instalación y configuración del XR Device Simulator para poder probar la aplicación de realidad virtual sin tener que utilizar dispositivos físicos de realidad virtual.

Finalmente, debido al tipo de juego, se le ha limitado el movimiento y el rango de visión al usuario en ciertas escenas. En los menús, el usuario no podrá desplazarse pero si que podrá rotar la cámara en todas las direcciones. En el caso del juego, como el avatar corre hacia delante, se le ha bloqueado tanto el movimiento como la rotación de la cámara, de esta forma se evitarán distracciones. Para ello en el *GameObject XR Origin* se ha creado una cámara con un componente denominado *Tracked Pose Driver*, en él se puede seleccionar el tipo de seguimiento. En los menús se seleccionará el tipo *Rotation Only* y en el juego se eliminará dicho componente ya que no queremos ningún tipo de seguimiento.

Además, en todos los *Canvas* creados para poder adaptarlos a la realidad virtual, se ha tenido que modificar el modo de renderizado a *World Space*. De esta forma el *Canvas* se quedará fijo en el entorno cuando el usuario rote la cámara.

Finalmente, tras implementar la realidad virtual el menú inicial sería como el mostrado en la Figura 8.7 con los controladores de la realidad virtual. En el resto de menús también se muestran los controladores con esos *Prefabs* pero durante la carrera se han eliminado ya que no hay nada interactuable, únicamente se utiliza el controlador izquierdo para cambiar de carril.

8.15.1. Compatibilidad con el metaverso de Meta

Una vez integrada la realidad virtual en el proyecto se continuó con las configuraciones necesarias para su compatibilidad con el metaverso de Meta. Dicha configuración se va a llevar a cabo teniendo en cuenta que no se dispone de equipos físicos de realidad virtual de Meta, como pueden ser las Meta Quest. Esto significa que no se realizará el despliegue final en dichos dispositivos sino que simplemente se va a garantizar su compatibilidad con Meta.

Figura 8.7: Menú inicial interactuable con los controladores de realidad virtual.

Primero se comenzará configurando el proyecto para que sea compatible con Android instalando el módulo de soporte de compilación de Android [86], que incluye el Android SDK, el NDK Tools y el OpenJDK. Para su instalación se debe abrir Unity Hub y en la pestaña *Install* hacer click en el icono de los ajustes y seleccionar *Add modules*.

Posteriormente se ha instalado el Meta XR Core SDK que permite la integración de Oculus [4] y se ha configurado como plataforma objetivo, Android ya que es la que utiliza Oculus Quest y Quest 2. Para ello en la pestaña *File* se debe seleccionar *Build Settings* y a continuación Android, una vez seleccionado hacer click en el botón *Switch Platform*.

Los siguientes pasos se han realizado en el apartado de *Player* que se encuentra tras seleccionar *Edit* =>*Project Settings* =>*Player* utilizando como guía [52].

- ***Company Name*:** Se ha indicado la UVa.
- ***Product Name*:** SugarRace.
- **Apartado de identificación:**
  - ***Package Name*:** Generalmente este nombre está compuesto por el nombre de la empresa y del producto, en este caso com.UVa.SugarRace.
  - ***Minimum API Level*:** Se ha indicado la versión de Android 10 (API level 29).
  - ***Target API Level*:** Se ha seleccionado *Automatic (highest installed)*.
- **Apartado de configuración:**
  - ***Scripting Backend list*:** Seleccionar IL2CPP.
  - Se ha dejado sin seleccionar el *checkbox* **ARMv7** y se ha seleccionado el **ARM64**.
  - ***Install Location list*:** Seleccionar *Automatic*.

- **Apartado de renderizado:**
  - ***Color Space*:** Seleccionar *Linear*.
  - Se ha dejado sin seleccionar el *checkbox* ***Auto Graphics API*** y se ha seleccionado el ***Multithreaded Rendering***.

Una vez realizadas estas configuraciones en *Edit =>Project Settings =>*Oculus se ha seleccionado el *checkbox* ***Low Overhead Mode***. Luego, en el panel de navegación de la izquierda se ha seleccionado *Quality* y se han realizado los siguientes cambios:

- ***Pixel Light Count*:** Se ha modificado el número máximo de recuento de luz de píxeles en uno.
- ***Global Mipmap Limit list*:** Se ha seleccionado *Full Resolution*.
- ***Anisotropic Textures list*:** Se ha cambiado a la opción *Per Texture*.
- ***Anti Aliasing list*:** Se ha establecido el valor a 4x.
- Se ha dejado sin seleccionar el *checkbox* ***Soft Particles***.
- Se ha seleccionado los *checkboxs* ***Realtime Reflections Probes*** y ***Billboards Face Camera Position***.

Por último, abrimos *Project Settings* y seleccionamos en el menú izquierdo Meta XR, aquí aparecerán una serie de recomendaciones como se muestra en la Figura 8.8 que al dar al botón *fix* se solucionarán.

Una vez se han completado todos los elementos de la *checklist*, como se puede ver en la Figura 8.9 ya esta el proyecto listo para ser ejecutado en unas gafas Meta Quest o Meta Quest 2.

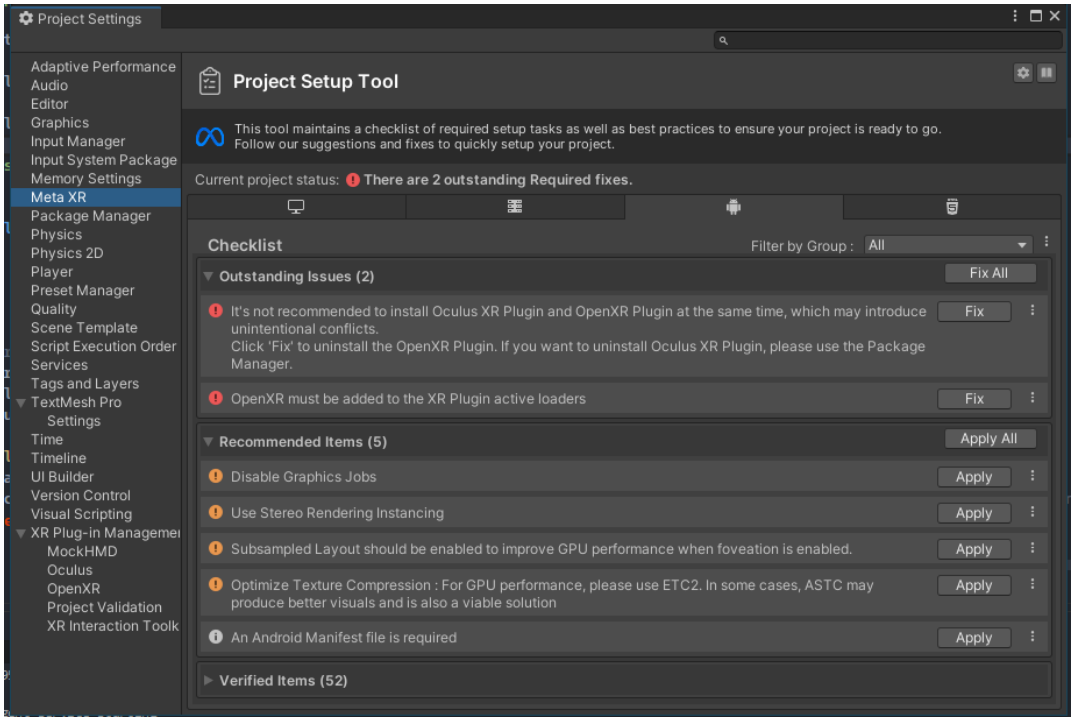

Figura 8.8: Ventana *Project Settings* al seleccionar la pestaña Meta XR.

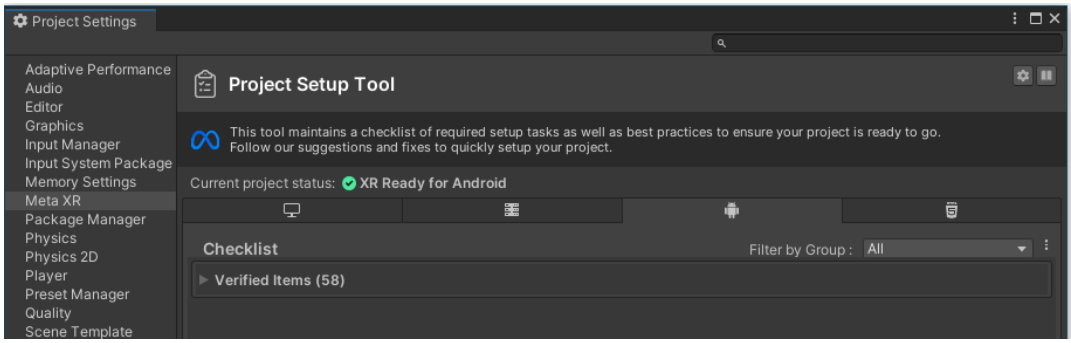

Figura 8.9: Ventana *Project Settings* al seleccionar la pestaña Meta XR tras darle a los botones de *fix*.

Capítulo 9

Pruebas

Este capítulo abordará los diferentes tipos de pruebas que han sido realizadas para comprobar el funcionamiento del videojuego. Se comenzará con las pruebas unitarias y de integración, comprobando que funcione cada unidad de código por separado y entre varios componentes del juego juntos.

Posteriormente, se comentarán las pruebas de rendimiento donde se evalúa cómo se comporta el juego. Además, se harán pruebas funcionales, realizadas por el tutor comprobando que las funcionalidades del juego se comporten según lo esperado y, por último, las pruebas de usabilidad y jugabilidad, donde se evaluará la experiencia de los usuarios tanto para recibir retroalimentación de la interfaz como de la mecánica y diversión al jugar el videojuego.

9.1. Pruebas unitarias y de integración

9.1.1. Pruebas unitarias

Respecto a las pruebas unitarias no han podido ser implementadas debido a las restricciones de tiempo del TFG. Su principal función es comprobar el funcionamiento de cierta parte del código de forma aislada. Un ejemplo de este tipo de test a nivel de *UseCase* sería el que se muestra en la Tabla 9.1.

Para abarcar los diferentes casos del algoritmo del cálculo del nivel de glucosa inicial habría que ir modificando los diferentes parámetros que afectan al nivel de glucosa, en este caso se ha realizado un ejemplo modificando el número de raciones de absorción lenta con un tiempo de 15 minutos.

En la Tabla 9.2 se muestran los casos de prueba más importantes, ya que son muchos y muy similares y en formato tabla alargaría demasiado la memoria. De esta forma cubriríamos tanto los test de caja negra como los de caja blanca. Los valores elegidos del tiempo

PU01	
Título	Cálculo del nivel de glucosa inicial al tomar alimentos.
Descripción	El algoritmo del cálculo de glucosa inicial debe ser capaz de calcular el nivel de glucosa inicial al modificar las raciones de alimentos de absorción lenta tras introducir 0 insulinas aplicadas y 15 minutos de tiempo a esperar.
Parámetros	Número de raciones: 2
Pasos	1. Asignar el tipo de ración y calcular el nivel de glucosa. 2. Asignar el número de insulinas y calcular el nivel de glucosa. 3. Asignar el tiempo a esperar y calcular el nivel de glucosa. 4. Asignar el número de raciones y calcular el nivel de glucosa.
Resultado	Nivel de glucosa: 140

Tabla 9.1: Prueba Unitaria 1: Algoritmo nivel de glucosa inicial.

corresponden con el tiempo en el cual cada tipo de alimento y la insulina alcanzan el pico de efecto.

Para no alargar aún más la memoria, el resto de pruebas unitarias no serán descritas. Todas ellas se definirían de forma similar a la mostrada anteriormente cubriendo tanto los casos de prueba de éxito como los fallidos si les hubiera. Además, habría que tener en cuenta las pruebas de caja negra y caja blanca como se ha hecho en el ejemplo anterior.

9.1.2. Pruebas de integración

Las pruebas de integración verifican que los diferentes módulos o componentes de la aplicación funcionan correctamente al probarse juntos. En este proyecto los componentes corresponden a cada una de las capas descritas en la Sección 7.1, por lo que estas pruebas verifican que la comunicación entre las diferentes capas sea efectiva.

Por ejemplo, habría que verificar que la comunicación entre el *script ChooseMenuPresenter* y el *script ChooseMenuUseCase* se haga correctamente comprobando también el envío del evento si procede. También sería necesario crear una prueba entre los *scripts* de la vista con el *script ChooseMenuPresenter*.

En los casos como el menú inicial y el del fin de nivel se podría crear un solo test de integración que verifique la comunicación desde la capa *View* hasta la capa *UseCase* pasando por la capa *Presenter*. Esto se debe a que estos casos de uso son más sencillos y se podría probar lo mismo en un único test.

Tipo	Raciones	Insulinas	Tiempo	Resultado
Lenta	0	0	0	90
Moderada	0	0	0	90
Rápida	0	0	0	90
Lenta	1	0	0	90
Moderada	1	0	0	90
Rápida	1	0	0	90
Lenta	0	1	0	90
Moderada	0	1	0	90
Rápida	0	1	0	90
Lenta	0	0	15	90
Moderada	0	0	15	90
Rápida	0	0	15	90
Lenta	1	0	15	115
Moderada	1	0	15	140
Rápida	1	0	15	190
Lenta	0	1	15	87
Moderada	0	1	15	87
Rápida	0	1	15	87
Lenta	1	1	15	112
Moderada	1	1	15	137
Rápida	1	1	15	187
Lenta	1	0	30	140
Moderada	1	0	30	190
Rápida	1	0	30	90
Lenta	0	1	30	85
Moderada	0	1	30	85
Rápida	0	1	30	85
Lenta	1	1	30	135
Moderada	1	1	30	185
Rápida	1	1	30	85
Lenta	1	0	60	190
Moderada	1	0	60	90
Rápida	1	0	60	90
Lenta	0	1	60	80
Moderada	0	1	60	80
Rápida	0	1	60	80
Lenta	1	1	60	180
Moderada	1	1	60	80
Rápida	1	1	60	80

Tabla 9.2: Diferentes casos de prueba del cálculo del nivel de glucosa inicial partiendo de un nivel de glucosa de 90.

9.2. Comparación hardware

En cuanto a capacidades de computación, ambos modelos de gafas cuentan con muchas más capacidades. Concretamente ambas cuentan con el doble de núcleos en sus procesadores y con una GPU dedicada mientras que el portátil utilizado solo cuenta con 4 núcleos y sin GPU dedicada.

Respecto a la RAM y el espacio de almacenamiento interno, es cierto que el ordenador cuenta con mucha más cantidad de ambas pero esto no es necesariamente una ventaja debido a que el ordenador no es un dispositivo que únicamente se centra en renderizar videojuegos al contrario de lo que sucede con las gafas. Además, se han realizado diferentes controlando la memoria utilizada durante diferentes ejecuciones del videojuego en el equipo de desarrollo y se ha comprobado que el juego nunca supera el 1Gb de RAM.

Finalmente, ambos modelos de gafas son capaces de producir una mejor resolución a una mayor frecuencia de actualización gracias a lo comentado anteriormente respecto a las capacidades de computación.

Tras esta comparación técnica de las especificaciones de dichos dispositivos se concluye que si el videojuego es capaz de ejecutarse sin problemas aparentes durante un periodo de tiempo en el ordenador, entonces será capaz de ejecutarse en los dispositivos de realidad virtual, ya que cuentan con mejores características.

En la Tabla 9.3 se muestra la comparativa explicada anteriormente entre las gafas Oculus Quest, Meta Quest 2 y el equipo donde se ha desarrollado.

	Oculus Quest	**Meta Quest 2**	**Ordenador**
Velocidad de la CPU	8 x 2,175GHz	8 X 2,1625GHz	4 x 2,30 GHz
GPU	Adreno 540	Adreno 650	–
RAM	4,0 Gb	6,0 Gb	16,0 GB
Almacenamiento interno	256Gb	128Gb	421Gb
Resolución	2880 x 1600 px	3664 x 1920 px	1920 x 1080 px
Frecuencia de actualización	70Hz	90Hz	60Hz

Tabla 9.3: Comparación de las especificaciones técnicas. [69], [97], [104].

9.3. Pruebas funcionales

Las pruebas funcionales tienen como objetivo comprobar que se estén cumpliendo todos los requisitos funcionales del videojuego, descritos en la Subsección 6.1.1. A continuación se muestran las pruebas que se han llevado a cabo.

PF01	
Título	Selección de nivel.
Descripción	El usuario modificará el nivel que se encuentra en el menú inicial.
Precondición	Encontrarse en la escena del menú inicial.
Pasos	1. El usuario pulsa sobre la flecha de la derecha.
Resultado	El nivel debe modificarse al número 2 y el panel izquierdo debe mostrar la información de dicho de nivel.

Tabla 9.4: Prueba funcional 1: Selección de nivel.

PF02	
Título	Pulsar el botón comenzar.
Descripción	El usuario pulsa el botón comenzar que se encuentra en el menú inicial.
Precondición	Encontrarse en la escena del menú inicial.
Pasos	1. El usuario pulsa sobre el botón de comenzar.
Resultado	La escena debe cambiar a la del menú de cálculo de nivel de glucosa inicial y si el nivel seleccionado es el 1 o el 2 el nivel de glucosa que se indica debe ser 90, si es el 3, 4 o 5 debe ser un valor múltiplo de 5 aleatorio entre 50 y 200.

Tabla 9.5: Prueba funcional 2: Pulsar el botón comenzar.

En la prueba de la Tabla 9.9 se ha englobado toda la lógica correspondiente a la carrera del juego para simplificar pero en realidad deberían crearse diferentes pruebas probando cada uno de los niveles, recolectando cada uno de los objetos y comprobando como se modifican los puntos y el nivel de glucosa.

PF03	
Título	Cálculo del nivel de glucosa inicial niveles 1, 2 y 3.
Descripción	El usuario pulsa los diferentes botones de la escena para modificar su nivel de glucosa inicial.
Precondición	Seleccionar el nivel 1, 2 o 3 y encontrarse en la escena del menú de cálculo de glucosa inicial.
Pasos	1. El usuario pulsa sobre los diferentes botones que aparecen en la escena.
Resultado	EL nivel de glucosa que aparece en la parte inferior se recalcula en función de los valores que ha modificado el usuario.

Tabla 9.6: Prueba funcional 3: Cálculo del nivel de glucosa inicial niveles 1, 2 y 3.

PF04	
Título	Cálculo del nivel de glucosa inicial nivel 4 y 5.
Descripción	El usuario pulsa los diferentes botones de la escena para modificar su nivel de glucosa inicial.
Precondición	Seleccionar el nivel 4 o 5 y encontrarse en la escena del menú de cálculo de glucosa inicial.
Pasos	1. El usuario pulsa sobre los diferentes botones que aparecen en la escena.
Resultado	EL nivel de glucosa que aparece en la parte inferior **no** se recalcula en función de los valores que ha modificado el usuario.

Tabla 9.7: Prueba funcional 4: Cálculo del nivel de glucosa inicial nivel 4 y 5.

PF05	
Título	Pulsar el botón jugar.
Descripción	El usuario pulsa el botón jugar que se encuentra en el menú de decisiones iniciales.
Precondición	Encontrarse en la escena del menú de decisiones iniciales.
Pasos	1. El usuario pulsa sobre el botón de jugar.
Resultado	La escena debe cambiar a la del juego donde el avatar comienza corriendo automáticamente y donde el nivel de glucosa debe ser el calculado en la escena anterior.

Tabla 9.8: Prueba funcional 5: Pulsar el botón jugar.

PF06	
Título	Ejecución de la carrera.
Descripción	El usuario es capaz de mover al personaje y recolectar objetos en los todos los niveles existentes.
Precondición	Encontrarse en la escena de la carrera.
Pasos	1. El usuario se mueve por las diferentes vías. 2. El usuario recolecta todos los tipos diferentes de objetos. 3. El usuario atraviesa un *checkpoint*.
Resultado	El resultado de esta prueba debe ser comprobado siguiendo las explicaciones dadas en la Sección 6.5.

Tabla 9.9: Prueba funcional 6: Ejecución de la carrera.

PF07	
Título	Pulsar el botón de volver al menú principal.
Descripción	El usuario debe ver la información relativa al nivel jugado y volver al menú principal.
Precondición	Encontrarse en la escena del menú de fin de nivel.
Pasos	1. El usuario mira la información mostrada. 2. El usuario pulsa el botón de volver al menú principal.
Resultado	La información mostrada sobre la frase informativa, el tiempo en rango y los puntos debe ser coherente respecto al nivel que se haya jugado y al pulsar el botón, la escena debe cambiar a la del menú inicial con el nivel 1 por defecto.

Tabla 9.10: Prueba funcional 7: Pulsar el botón de volver al menú principal.

9.4. Pruebas de usabilidad y jugabilidad

Por último, se han realizado pruebas de usabilidad y jugabilidad a diferentes usuarios voluntarios. Dichas pruebas tienen como objetivo conseguir retroalimentación sobre la interfaz de usuario y el nivel de diversión del juego. De esta forma el desarrollador consigue opiniones de las personas que van a utilizar el juego, pudiendo así solucionar posibles problemas con la interfaz antes de que el juego sea publicado o conseguir nuevas ideas para futuras implementaciones del juego relacionadas con la jugabilidad.

En el caso de este proyecto se genero un ejecutable del juego considerado ya la primera versión del mismo y se realizaron pruebas con usuarios finales, en este caso amigos y familiares de la alumna. A cada usuario se le dio una breve explicación del objetivo y funcionamiento del juego. Al no tener dispositivos físicos de realidad virtual, las pruebas se realizaron en un ordenador.

La prueba se llevo a cabo con 5 usuarios de los cuales 4 realizaron la prueba con éxito. El quinto tuvo algunos problemas relacionados con los controladores de la realidad virtual ya que al no tenerlos se utiliza un simulador. Ya que es un problema con el simulador se puede concluir que el juego es intuitivo y no existen problemas aparentes en la actualidad si se ejecuta en unos dispositivos de realidad virtual.

En cuanto a la jugabilidad, los usuarios estaban de acuerdo que es mejorable pero todos coinciden que en cuanto a la parte educativa cumple satisfactoriamente su función. Dado que el objetivo del juego es la educación diabetológica es un resultado muy positivo aunque la parte de jugabilidad se podría mejorar en un futuro.

Capítulo 10

Seguimiento del proyecto

Este capítulo se centra en explicar cómo ha sido el desarrollo del proyecto. Tal y como ya se ha explicado en el Capítulo 5, se ha usado Scrum como marco de gestión de trabajo de metodología ágil y por ello este capítulo va a estar dividido en *sprints*. En cada uno de ellos habrá una tabla con los siguientes apartados:

- **Historia de usuario** (H.U.) que está asociada a la *issue*.
- ***Issue*:** Título descriptivo de la tarea.
- **Tiempo estimado:** Tiempo que se estimó que iba a durar la *issue* en la planificación inicial.
- **Tiempo empleado:** Tiempo real que se ha tardado en realizar la *issue*.
- **Estado** de la *issue* al terminar el *sprint*. Puede ser:
  - ***Open:*** *Issue* del *Product Backlog*.
  - ***Planned:*** *Issue* que se va a realizar en este *sprint*.
  - ***Doing:*** *Issue* que se está haciendo.
  - ***Blocked:****Issue* que no se puede continuar haciendo.
  - ***Closed:*** *Issue* terminada.

Para el seguimiento del proyecto se ha utilizado la herramienta Trello, la cual facilita la planificación y clasificado de las tareas en los *sprints*.

10.1. Sprint 0 (05/12/2023 - 15/02/2024)

Comenzó con una reunión con el tutor de HP-SCDS donde se expuso el objetivo y motivación del proyecto junto con una explicación sobre conceptos y definiciones importantes sobre

la diabetes. También, se vieron los diferentes requisitos que la empresa HP-SCDS espera que se cumplan y unas breves explicaciones de cómo podrían ser los algoritmos del cálculo de glucosa inicial, cálculo de glucosa a lo largo de la carrera y cálculo de puntos.

Este Sprint 0 se ha enfocado a aprender sobre las nuevas tecnologías que se usan en el proyecto, como el motor de desarrollo Unity o el lenguaje C#. Además, a partir de la idea inicial del proyecto, se ha creado una primera versión del *Product Backlog* que se fue completando a lo largo de los *sprints* dependiendo de las necesidades que surjan. Se investigó sobre las arquitecturas que se podrían utilizar además de un análisis de los requisitos de HP-SCDS para poder crear el *Product Backlog*. Las horas invertidas en este sprint han sido **40 horas**.

Cabe destacar que este es el único *sprint* que no dura 3 semanas debido a que en enero se celebraba la convocatoria ordinaria y extraordinaria de exámenes del primer cuatrimestre y por ello si que se empezó con la formación en las nuevas tecnologías en los ratos libres pero no se empezó con el desarrollo hasta que no finalizaran los exámenes.

En la reunión de fin de *sprint* se resolvieron todas las dudas sobre el proyecto, se realizó la calendarización inicial y se llevo a cabo la planificación del Sprint 1, donde se estableció que en dicho *sprint* se realizaría la introducción y el estado del arte respecto a la memoria y en cuanto al desarrollo, la configuración del entorno de desarrollo en Unity, búsqueda e implementación de recursos para el entorno del juego y la creación del *script* que debe permitir moverse al usuario.

10.2. Sprint 1 (15/02/2024 - 14/03/2024)

En este *sprint* se completaron todas las historias de usuario que se planificaron. En la Tabla 10.1 podemos ver todas las tareas realizadas.

Según la planificación, este sprint debió finalizar el día 7 de marzo y tener una carga de trabajo de 41,5 horas pero, debido a que el tutor de HP-SCDS se fue de vacaciones sin previo aviso no se pudo realizar la reunión de inicio/fin de sprint. Por ello se decidió alargar una semana este sprint dedicando esta semana de más a la realización de los diagramas de análisis y diseño del proyecto, por esta razón no existe una planificación de estas dos tareas en la Tabla 10.1.

Además de dicha desviación, uno de los riesgos se materializó debido a que no se encontraron recursos gratuitos para simular el entorno del juego, que en un primer momento se estableció que sería una pista de atletismo donde el avatar iría corriendo por una de las calles y donde aparecería comida e insulina por las diferentes calles para que el usuario tomara la decisión de tomarlo o no. Debido a que todos los recursos fiables eran de pago, se tomó la decisión de cambiar el escenario. Tras encontrar opciones similares con la misma dinámica de los carriles, se decidió implementar *assets* (recursos) que simulan tres vías de tren, haciendo un símil a las calles de la pista de atletismo.

Una vez resuelto este problema se llevó a cabo su implementación en Unity y la creación del *script* para que el usuario pueda moverse entre las diferentes vías del tren.

H.U.	Issue	Tiempo Est.	Tiempo Empl.	Estado
HU1	Configuración del entorno de desarrollo.	2 horas	1,5 horas	Closed
HU2	Encontrar recursos para simular una pista de atletismo en Unity.	5 horas	7 horas	Closed
HU3	Implementación de los recursos que simulan el entorno del juego.	5 horas	4 horas	Closed
HU4	Añadir *script* con los movimientos del usuario.	5 horas	4 horas	Closed
-	Creación de los diagramas de análisis.		12 horas	Closed
-	Creación de los diagramas de diseño.		13 horas	Closed
-	Redacción de la sección de seguimiento correspondiente a este *sprint*	30 minutos	30 minutos	Closed
-	Redacción de la sección de implementación correspondiente a este *sprint*	2 horas	2,5 horas	Closed
-	Redacción del Capítulo 1	14 horas	15 horas	Closed
-	Redacción del Capítulo 2	8 horas	9 horas	Closed
Total		41,5 horas	68,5 horas	10/10

Tabla 10.1: *Sprint Backlog* del Sprint 1.

En paralelo se llevó a cabo la redacción del capítulo de introducción y del estado del arte de la memoria en los que se tardó más tiempo del estimado por la falta de soltura en la redacción de documentos.

En la reunión de retrospectiva se comentaron los aspectos positivos y negativos del *sprint*. Entre los positivos se encuentra la extensa documentación y tutoriales sobre Unity lo que ha tenido como consecuencia que su desarrollo se realizara en menos tiempo del estimado. Además, se destacó que se habían completado todos los objetivos marcados para este *sprint*, lo que denota una buena gestión de la cantidad de trabajo. Por otra parte, como aspecto negativo se encuentra la falta de práctica en la redacción.

Para el *sprint* siguiente, se ha planificado realizar algunas tareas de las historias de usuario 5, 6, 7, 8, 9 y 11 además de la redacción del capítulo de planificación.

10.3. Sprint 2 (14/03/2024 - 04/04/2024)

En este *sprint* también se completaron todas las tareas que se habían planificado, como se puede ver en la Tabla 10.2.

En esta ocasión se materializó otra vez el mismo riesgo pero esta vez debido a que no se encontraron recursos que simularan una pluma[1]. Este *asset* es el que iba a simular la insulina dentro del juego pero al no encontrar recursos se sustituyó por un pequeño frasco.

[1]Dispositivo utilizado para la auto-administración de la insulina

H.U.	Issue	Tiempo Est.	Tiempo Empl.	Estado
HU5	Implementación del menú inicial.	1 hora	1 hora	Closed
	Implementación de selección de nivel dentro del menú inicial.	2 horas	3 hora	Closed
	Implementación de la pantalla de resumen de nivel.	3 horas	2,5 horas	Closed
HU6	Encontrar recursos para simular alimentos e insulina en Unity.	3 horas	4 horas	Closed
HU7	Generar el circuito.	2 horas	3 horas	Closed
	Introducir un fin de nivel.	1 hora	30 minutos	Closed
	Generación de los alimentos e insulina a lo largo del circuito.	2 hora	4 horas	Closed
HU8	Recolección de alimentos e insulina.	2 horas	2 horas	Closed
HU9	Encontrar recursos para la barra de glucosa y su implementación.	2 horas	3 horas	Closed
HU11	Implementar el algoritmo de suma/resta de glucosa del nivel 1.	2 horas	3 horas	Closed
-	Redacción de la sección de seguimiento correspondiente a este *sprint*	30 minutos	30 minutos	Closed
-	Redacción de la sección de implementación correspondiente a este *sprint*	6 horas	8 horas	Closed
-	Redacción del Capítulo 5	15 horas	14,5 horas	Closed
Total		41,5 horas	49 horas	13/13

Tabla 10.2: *Sprint Backlog* del Sprint 2.

En cuanto a los recursos de la barra de glucosa sucedió algo parecido pero en este caso la desarrolladora creó su propia barra de glucosa. Como consecuencia se tardó más de lo que se había estimado.

Además la generación de los alimentos e insulina se hizo una estimación optimista debido a que la idea inicial era reutilizar el *script* de generación del circuito con unas pequeñas modificaciones. Estas modificaciones tenían una complejidad mucho mayor de lo esperado y por ello se empleó mucho más tiempo del estimado.

En la reunión de retrospectiva se llegó a la conclusión de que se ha mejorado en la parte de redacción, debido a que se ha tardado menos de lo que se había estimado. Para el siguiente *sprint* se realizarán algunas de las tareas de las historias de usuario 3, 10 y 11 además de redactar la parte de análisis.

10.4. Sprint 3 (04/04/2024 - 25/04/2024)

Las tareas realizadas durante este *sprint* se pueden ver en la Tabla 10.3:

H.U.	Issue	Tiempo Est.	Tiempo Empl.	Estado
HU03	Implementación de *checkpoints* al mapa.	2 horas	2 horas	Closed
HU10	Implementar la suma de puntos del usuario.	3 horas	3 horas	Closed
HU11	Implementar el algoritmo de suma/resta de glucosa del nivel 2.	6 horas	8 horas	Closed
-	Redacción de la sección de seguimiento correspondiente a este *sprint*	30 minutos	30 minutos	Closed
-	Redacción de la sección de implementación correspondiente a este *sprint*	3 horas	3 horas	Closed
-	Redacción del Capítulo 6	20 horas	20 horas	Closed
Total		34,5 horas	36,5 horas	6/6

Tabla 10.3: *Sprint Backlog* del Sprint 3.

Durante este *sprint* se implemento el segundo algoritmo que se utilizará para el nivel 2. Al ser el segundo algoritmo hubo que implementar el patrón estrategia donde hubo una serie de problemas y por ello se tardó más.

Durante la reunión de retrospectiva se resaltó la buena estimación para las tareas de redacción y que se habían completado todas las tareas. También se vio que el código según crecía estaba demasiado acoplado a Unity, por lo que se propuso una refactorización del código para seguir de una manera más estricta la *Simple Clean Architecture*.

Esta refactorización junto con la implementación del resto de niveles y añadir el funcionamiento de los *checkpoints*, son las tareas que se realizarán durante el siguiente *sprint* además de la redacción del capítulo de diseño incluyendo los diagramas creados en el Sprint 1.

10.5. Sprint 4 (25/04/2024 - 16/05/2024)

Las tareas realizadas durante este *sprint* se pueden ver en la Tabla 10.4:

Las implementaciones de los diferentes niveles se han podido realizar en tan solo 4 horas gracias a que ya se había implementado el patrón estrategia en el Sprint 3, consiguiendo así que el añadir nuevos niveles no sea algo complicado de desarrollar. Además, para que aparezcan los diferentes objetos en cada uno de los niveles, se ha creado otra tarea que aparece en la primera fila de la Tabla 10.4.

En cuanto a la retrospectiva de este sprint no se ha comentado nada relevante, el sprint ha ido mejor de lo que se esperaba tardando menos tiempo del estimado, lo cual es algo muy positivo.

Para el Sprint 5 se ha planificado implementar al avatar del juego, incluir un panel infor-

H.U.	Issue	Tiempo Est.	Tiempo Empl.	Estado
HU07	Creación de diferentes *pool* por cada nivel	5 horas	5 horas	Closed
HU11	Implementación nivel 3	2 horas	2 horas	Closed
HU11	Implementación nivel 4	2 horas	1 hora	Closed
HU11	Implementación nivel 5	2 horas	1 hora	Closed
HU10	Funcionamiento de *checkpoints*	2 horas	2 horas	Closed
-	Refactorización del código.	9 horas	10 horas	Closed
-	Redacción de la sección de implementación correspondiente a este *sprint*	3 horas	3 horas	Closed
-	Redacción de la sección de seguimiento correspondiente a este *sprint*	30 minutos	30 minutos	Closed
-	Redacción del Capítulo 7	20 horas	18 horas	Closed
Total		45,5 horas	42,5 horas	9/9

Tabla 10.4: *Sprint Backlog* del Sprint 4.

mativo de los diferentes objetos que aparecen en cada nivel y la configuración y adaptación del proyecto a la realidad virtual. En cuanto a la memoria, se ha planificado redactar y repasar el capítulo de implementación además de redactar el seguimiento de ese *sprint*.

10.6. Sprint 5 (16/05/2024 - 06/06/2024)

En este *sprint* también se completaron todas las tareas que se habían planificado, como se puede ver en la Tabla 10.5.

En este *sprint* se ha tardado más de lo planificado debido a que durante la configuración y adaptación de la realidad virtual se añadió un componente a un *GameObject* que desencadenó varios problemas en el funcionamiento de esta tecnología, perdiendo así varias horas en solucionarlo.

En cuanto a la tarea de añadir el panel informativo, se han estimado 6 horas debido a que en realidad son 5 paneles, uno por cada nivel. Esto se debe a que en cada nivel se van añadiendo diferentes objetos que se pueden recolectar y no es posible hacer uno para todos. Por ello había que crear los *sprites*[2] de los objetos que no lo tuvieran y añadirlo a cada uno de los paneles, además de crear el *script* que controle que panel debe aparecer en cada momento.

Durante la reunión de retrospectiva se destacó que, aunque se ha dedicado más tiempo del estimado, se ha conseguido finalizar todas las tareas. Para el Sprint 6 se planificó la redacción del capítulo de pruebas, conclusiones y los manuales, además de redactar el seguimiento de dicho sprint y repasar la memoria para su finalización.

[2]Objeto gráfico 2D

H.U.	Issue	Tiempo Est.	Tiempo Empl.	Estado
HU03	Implementación del avatar	5 horas	5 horas	Closed
HU05	Añadir panel informativo en el menú inicial	6 horas	6 horas	Closed
HU13	Configuración de la realidad virtual	10 horas	13 horas	Closed
HU13	Adaptación de los Canvas y controles a la VR	7 horas	9 horas	Closed
HU13	Compatibilidad con el metaverso	6 horas	6 horas	Closed
-	Redacción del apartado de realidad virtual	3 horas	3 horas	Closed
-	Redacción del apartado del metaverso	2 horas	2 horas	Closed
-	Redacción de la sección de implementación correspondiente a este *sprint*	5 horas	4 horas	Closed
-	Repaso del Capítulo 8	2 horas	2 horas	Closed
-	Redacción de la sección de seguimiento correspondiente a este *sprint*	30 minutos	30 minutos	Closed
Total		46,5 horas	50,5 horas	10/10

Tabla 10.5: *Sprint Backlog* del Sprint 5.

10.7. Sprint 6 (06/06/2024 - 20/06/2024)

En este sprint se han realizado las tareas mostradas en la Tabla 10.6:

H.U.	Issue	Tiempo Est.	Tiempo Empl.	Estado
-	Redacción del Capítulo 9	12 horas	11 horas	Closed
-	Redacción del Capítulo 11	6 horas	6 horas	Closed
-	Redacción del Apéndice A	2 horas	2 horas	Closed
-	Redacción del Apéndice B	2 horas	2 horas	Closed
-	Redacción de la sección de seguimiento correspondiente a este *sprint*	30 minutos	15 minutos	Closed
-	Repaso de la memoria.	12 horas	12 horas	Closed
Total		34,5 horas	33,25 horas	6/6

Tabla 10.6: *Sprint Backlog* del Sprint 6.

Durante este *sprint* no hubo ningún problema a destacar e incluso se terminó en menos tiempo del estimado. Con este *sprint* se dio por finalizado este trabajo de fin de grado.

10.8. Comparación con el tiempo estimado

En la Sección 5.2 se realizó la planificación inicial donde se establecieron las diferentes fechas aproximadas de inicio y fin de los *sprints* además de una estimación orientativa de 300 horas para la realización del proyecto divididas entre los diferentes *sprints*.

En cuanto a las fechas planificadas ha surgido un desvío en el Sprint 1 explicado en la Sección 10.2, por ello dicho sprint ha tenido una semana más de la planificada y como consecuencia el último sprint una menos. Debido a esto las horas empleadas para Sprint 1, 68,5 horas, sobrepasa lo que se había estimado inicialmente, 50 horas. Teniendo en cuenta que ha sido una semana más sería lo proporcional y compensa con el Sprint 6 donde se han empleado 33,5 horas en vez de las 40 horas que se había estimado, respetando así las fechas de inicio y fin del proyecto a pesar de esa variación.

En el resto de *sprints* la variación entre lo que se estimó en la Sección 5.2 y las horas que realmente se han empleado ha sido menor a 5 horas por lo que no supone una diferencia muy significativa. Si sumamos todos los tiempos estimados en esta sección y los tiempos empleados obtenemos un tiempo estimado de 284 horas y 320,25 horas de tiempo empleado al proyecto, satisfaciendo así las 300 horas que se indica en la guía docente del trabajo de fin de grado.

Capítulo 11

Conclusiones y líneas futuras

11.1. Conclusiones

El objetivo del trabajo de fin de grado radica en crear un videojuego 3D, enfocado a personas debutantes en diabetes (principalmente niños y adolescentes), orientado al metaverso y en realidad virtual con el fin de educar de una forma divertida sobre esta enfermedad. Tras varios meses trabajando en ello, el objetivo se ha conseguido alcanzar de manera exitosa utilizando Unity y Visual Studio como herramientas de desarrollo del videojuego cumpliendo así con todos los requisitos.

A pesar de las dificultades encontradas al buscar recursos gratuitos, se ha conseguido implementar una interfaz de usuario coherente e intuitiva, además de ser adecuada para el público objetivo.

En cuanto a la lógica del juego, se ha conseguido desarrollar algoritmos coherentes y que facilitan el entendimiento de muchos conceptos relacionados con la diabetes a aquellas personas que acaban de debutar en esta enfermedad.

Respecto a la planificación del proyecto, exceptuando algunos sucesos inesperados, generalmente ha sido la esperada y se ha conseguido terminar el proyecto en la convocatoria ordinaria utilizando el marco de trabajo de metodología ágil Scrum en un proyecto de una duración mayor a los que se realizan durante el grado.

En cuanto a los objetivos docentes y personales, también se han conseguido alcanzar. Se han podido aplicar multitud de conceptos y conocimientos adquiridos durante la carrera, además de ampliarlos con el uso de nuevas tecnologías como Unity o Visual Studio. También he podido sumergirme por primera vez en el mundo del desarrollo de videojuegos 3D en realidad virtual, consiguiendo así nuevas habilidades en este campo.

Además, gracias al desarrollo de este proyecto he adquirido conocimientos importantes sobre la diabetes para entender el estilo de vida que llevan estas personas, los desafíos que

enfrentan y las necesidades que aún quedan por cubrir. Y, aunque he conseguido desarrollar una herramienta que puede ser de gran utilidad para todas aquellas personas que tienen que tratar con esta enfermedad en su día a día, aún existe margen de mejora.

En resumen, el logro de los objetivos demuestra la eficacia y utilidad de este videojuego en el ámbito de la educación diabetológica. Aunque durante el camino se han encontrado algunas dificultades, el proyecto final cumple con las expectativas.

11.2. Líneas futuras

Tras haber realizado el videojuego, han surgido diferentes trabajos que podrías ser interesantes realizar para mejorar y ampliar este videojuego en el futuro.

- **Mejora del generador de objetos durante la carrera:** Actualmente en todos los niveles aparecen una insulina y un alimento a la misma altura dando la opción al usuario de elegir entre ellas o no recoger ningún objeto. Se propone mejorar esta funcionalidad para que en los últimos niveles se complique haciendo que a veces solo haya insulina o solo haya un tipo de alimento. Hay que tener en cuenta que no puede ser totalmente aleatorio ya que si no aparecen alimentos el usuario podría perder sin que fuera su culpa.
- **Añadir persistencia:** Debido a las restricciones de tiempo del trabajo de fin de grado se decidió no implementar bases de datos aunque si sería de utilidad en un futuro para guardar la información de los puntos o el tiempo en rango del usuario.
- **Implementación de más recursos como sonidos o nuevos mapas:** La implementación de sonidos al recolectar objetos o al pulsar ciertos botones sería una forma de dar retroalimentación al usuario de que ha ido todo correctamente. Además, para no hacer el videojuego muy monótono se podrían implementar más variedad de mapas.
- **Añadir nueva funcionalidad como variaciones en la velocidad:** Otra funcionalidad interesante que se podría añadir sería que el usuario pudiera modificar la velocidad y dependiendo de esta se restarán más o menos unidades a su nivel de glucosa. A más velocidad el ejercicio físico le restaría una mayor cantidad de unidades de glucosa, al igual que si reduce la velocidad se le restarán menos unidades.

Bibliografía

[1] 1000logos. OUTLOOK LOGO. https://1000logos.net/outlook-logo/. Accedido el 18-04-2024.

[2] 1000logos. VISUAL STUDIO LOGO. https://1000logos.net/visual-studio-logo/. Accedido el 18-04-2024.

[3] Asociación Navarra de Diabetes. Diabegame. https://anadi.es/actividades-diabetes-anadi/diabegame/. Accedido el 21-02-2024.

[4] Asset Store. Meta XR Core SDK. https://assetstore.unity.com/packages/tools/integration/meta-xr-core-sdk-269169. Accedido el 04-06-2024.

[5] Astah. Astah is a modeling tool. https://astah.net/products/. Accedido el 17-04-2024.

[6] Astah. Leverage the power of software modeling. https://astah.net/. Accedido el 17-04-2024.

[7] Astah. Team Licensing Options. https://astah.net/pricing/team/, 2024. Accedido el 24-03-2024.

[8] Boucher, A. DIABETES – MEDICAL VR GAME. https://sbanimation.com/case-studies/diabetes-voyager-medical-vr/. Accedido el 21-02-2024.

[9] Brazil Journal. Finalmente, a insulina da BIOMM vai sair do papel. https://braziljournal.com/finalmente-a-insulina-da-biomm-vai-sair-do-papel/, 2024. Accedido el 28-03-2024.

[10] Centros para el Control y la Prevención de Enfermedades. Te acaban de diagnosticar diabetes tipo 1. https://www.cdc.gov/diabetes/spanish/basics/diabetes-type-1-diagnosis.html, 2022. Accedido el 03-03-2024.

[11] Chen Ling, Ph.D., Shriram Seetharaman y Lubna Mirza. Roles of Serious Game in Diabetes Patient Education. https://journals.sagepub.com/doi/10.1177/10468781221120686?icid=int.sj-abstract.citing-articles.4, 2022. Accedido el 22-02-2024.

[12] Daniel Parra. Patrones de diseño – Object Pool. https://thepowerups-learning.com/patrones-de-diseno-object-pool/. Accedido el 05-05-2024.

[13] Daniel Parra. Patrones de diseño – Observer. https://thepowerups-learning.com/patrones-de-diseno-observer/. Accedido el 23-04-2024.

[14] Daniel Parra. Patrones de diseño – Service Locator. https://thepowerups-learning.com/patrones-de-diseno-service-locator/. Accedido el 13-05-2024.

[15] Daniel Parra. Patrones de diseño – Singleton y Monostate. https://thepowerups-learning.com/patrones-de-diseno-singleton-y-monostate/. Accedido el 23-04-2024.

[16] Daniel Parra. Patrones de diseño – Strategy. https://thepowerups-learning.com/patrones-de-diseno-strategy/. Accedido el 23-04-2024.

[17] Datlas, E. Roles en metodología SCRUM ¿Cuáles son y por qué son tan importantes? . https://blogdatlas.wordpress.com/2023/04/02/roles-en-metodologia-scrum-cuales-son-y-por-que-son-tan-importantes-manuales-datlas/, 2023. Accedido el 09-03-2024.

[18] Desire. Scrum. https://desire.webs.uvigo.es/contenidos/scrum/. Accedido el 09-03-2024.

[19] Diego Coder. Introducción a las "Clean Architectures". https://medium.com/@diego.coder/introducci%C3%B3n-a-las-clean-architectures-723fe9fe17fa, 2023. Accedido el 13-05-2024.

[20] Digital, S. DART, una app con realidad aumentada para motivar a los pacientes de diabetes a hacer deporte. https://www.consalud.es/saludigital/innovacion-tecnologica/dart-app-con-realidad-aumentada-motivar-pacientes-diabetes-hacer-deporte_139179_102.html, 2024. Accedido el 21-02-2024.

[21] Drumond, C. Qué es scrum y cómo empezar. https://www.atlassian.com/es/agile/scrum. Accedido el 08-03-2024.

[22] Esneca. Mitos y verdades sobre los alimentos ricos en hidratos de carbono. https://www.esneca.com/blog/alimentos-ricos-en-hidratos-de-carbono/, 2018. Accedido el 06-03-2024.

[23] Fernández, R. Juegos VR para móviles: qué necesitas para jugar y cuáles son los mejores. https://www.movilzona.es/noticias/juegos/mejores-juegos-vr-movil-accesorios/, 2023. Accedido el 29-02-2024.

[24] Food & Nutrition Magazine. Carb Counting with Lenny. https://foodandnutrition.org/may-2013/carb-counting-lenny/, 2013. Accedido el 22-02-2024.

[25] Formadores IT. ¿Qué es Gitlab y para qué sirve? https://formadoresit.es/que-es-gitlab-y-para-que-sirve/, 2024. Accedido el 17-04-2024.

[26] GaeaPeople. Hidratos de absorción rápida y lenta. https://www.solucionesparadiabetes.com/magazine-diabetes/hidratos-de-absorcion-rapida-y-lenta/, 2022. Accedido el 06-03-2024.

[27] Gamarra, G. Coste de un trabajador para la empresa. https://factorialhr.es/blog/coste-empresa-trabajador/, 2024. Accedido el 24-03-2024.

[28] Genki Sano. Simplified Clean Architecture Design Pattern for Unity. https://genki-sano.medium.com/simplified-clean-architecture-design-pattern-for-unity-967931583c47. Accedido el 29-04-2024.

[29] Genki Sano. Simplified Clean Architecture Design Pattern for Unity. https://genki-sano.medium.com/simple-clean-architecture-762b90e58d91. Accedido el 12-05-2024.

[30] Git. Control de versiones con Git. https://www.ticarte.com/contenido/control-de-versiones-con-git. Accedido el 17-04-2024.

[31] Git. Getting Started - What is Git? https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F. Accedido el 17-04-2024.

[32] GitHub. Acerca de GitHub Copilot para empresas. https://docs.github.com/es/enterprise-cloud@latest/copilot/copilot-business/about-github-copilot-business, 2024. Accedido el 24-03-2024.

[33] GitHub. Programación Reactiva. https://ferestrepoca.github.io/paradigmas-de-programacion/reactive/reactive_teoria/index.html, 2024. Accedido el 12-06-2024.

[34] GitHub. Where future developers meet. https://education.github.com/, 2024. Accedido el 24-03-2024.

[35] GitLab. About GitLab Behind the scenes of The DevSecOps Platform. https://about.gitlab.com/company/. Accedido el 17-04-2024.

[36] GLASSDOOR. Sueldos de Unity Developer. https://www.glassdoor.es/Sueldos/unity-developer-sueldo-SRCH_KO0,15.htm, 2024. Accedido el 24-03-2024.

[37] Goicochea, A. Análisis de los riesgos de un proyecto. https://anibalgoicochea.com/2012/11/23/analisis-de-los-riesgos-de-un-proyecto/, 2012. Accedido el 02-03-2024.

[38] Google VR. Google Cardboard. https://arvr.google.com/intl/es_es/cardboard/. Accedido el 16-06-2024.

[39] GR, R. Cómo funciona y en qué se usa la realidad virtual. https://www.adslzone.net/reportajes/tecnologia/realidad-virtual-rv/. Accedido el 20-02-2024.

[40] Ibarra, R. A más covid, más casos de diabetes tipo 1, pero nadie sabe por qué. https://www.abc.es/salud/enfermedades/covid-casos-diabetes-tipo-sabe-20230904171351-nt.html, 2023. Accedido el 18-02-2024.

[41] Innovación Digital 360. Qué son los cascos o gafas de realidad virtual y cómo funcionan. https://www.innovaciondigital360.com/i-a/que-son-los-cascos-o-gafas-de-realidad-virtual-y-como-funcionan/. Accedido el 04-06-2024.

[42] IONOS. El diagrama de casos de uso en UML. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-casos-de-uso/, 2020. Accedido el 30-03-2024.

[43] Javier Sáez Hurtado. Qué es el metaverso, ejemplos y cómo se accede. https://www.iebschool.com/blog/el-metaverso-origen-definicion-y-la-apuesta-de-facebook-tecnologia/, 2022. Accedido el 08-06-2024.

[44] Making Diabetes Easier. Glucozor, el juego que permite a los niños de 8 a 12 anños ser más conscientes de la diabetes y su tratamiento. https://www.makingdiabeteseasier.com/es/gestionando-la-diabetes/para-padres-e-hijos/glucozor. Accedido el 22-02-2024.

[45] María P. Russo, María F. Grande-Ratti ... Prevalencia de diabetes, características epidemiológicas y complicaciones vasculares. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10161833/, 2023. Accedido el 17-02-2024.

[46] Mateo Ball. The Metaverse: What It Is, Where to Find it, and Who Will Build It. https://www.matthewball.co/all/themetaverse, 2020. Accedido el 08-06-2024.

[47] Mauricio, A. ¿Cuál es la diferencia entre Casos de Usos y Escenarios? https://our-academy.org/posts/escenarios-y-casos-de-uso. Accedido el 30-03-2024.

[48] Mayo Clinic. Coma diabético. https://www.mayoclinic.org/es/diseases-conditions/diabetic-coma/symptoms-causes/syc-20371475, 2022. Accedido el 18-02-2024.

[49] Mayo Clinic. Hiperglucemia en la diabetes. https://www.mayoclinic.org/es/diseases-conditions/hyperglycemia/symptoms-causes/syc-20373631, 2022. Accedido el 18-02-2024.

[50] Medline Plus. Mitos y realidades acerca de la diabetes. https://medlineplus.gov/spanish/ency/patientinstructions/000964.htm, 2023. Accedido el 03-03-2024.

[51] Meidum. C# logo for stickers + stuff. https://medium.com/@chris.mckee/c-logo-for-stickers-stuff-bca39b31cba3. Accedido el 18-04-2024.

[52] Meta Quest. Configure Unity Settings. https://developer.oculus.com/documentation/unity/unity-conf-settings/. Accedido el 06-06-2024.

[53] Meta Quest. Meta XR Core SDK (UPM). https://developer.oculus.com/downloads/package/meta-xr-core-sdk/, 2024. Accedido el 10-06-2024.

[54] Metodologías Ágiles. How can you use user stories to identify technical spikes and research efforts? https://www.linkedin.com/advice/1/how-can-you-use-user-stories-identify-technical-8ppec?lang=en&originalSubdomain=es, 2024. Accedido el 23-03-2024.

[55] Metodologías Ágiles. Requerimientos funcionales y no funcionales en Scrum. https://metodologiasagiles.org/requerimientos-funcionales-y-no-funcionales-en-scrum/, 2024. Accedido el 30-03-2024.

[56] Microsoft. 7 cosas interesantes que no sabía acerca de Outlook. https://support.microsoft.com/es-es/office/7-cosas-interesantes-que-no-sab%C3%ADa-acerca-de-outlook-314cf31f-5cf9-4643-bc50-6d444107c6ae. Accedido el 17-04-2024.

[57] Microsoft. Introducción a Microsoft Teams. https://support.microsoft.com/es-es/office/introducci%C3%B3n-a-microsoft-teams-b98d533f-118e-4bae-bf44-3df2470c2b12. Accedido el 17-04-2024.

[58] Microsoft 365. Encontrar el plan de Microsoft 365 adecuado. https://www.microsoft.com/es-es/microsoft-365/business#heading-ocb6f5, 2024. Accedido el 24-03-2024.

[59] Microsoft Learn. Action<T>Delegate. https://learn.microsoft.com/es-es/dotnet/api/system.action-1?view=net-8.0. Accedido el 30-05-2024.

[60] Microsoft Learn. ¿Qué es Visual Studio? https://learn.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2022. Accedido el 15-04-2024.

[61] Mira, A. R. Realidad virtual en la educación: aprendizaje y juego. https://www.tokioschool.com/noticias/realidad-virtual-educacion/, 2020. Accedido el 03-03-2024.

[62] Mira, A. R. ¿Qué son las dinámicas de videojuegos? https://www.tokioschool.com/noticias/que-son-dinamicas-videojuegos/, 2020. Accedido el 01-03-2024.

[63] Mixamo. Mixamo. https://www.mixamo.com/#/. Accedido el 08-06-2024.

[64] Modesttree. Zenject. https://github.com/modesttree/Zenject. Accedido el 05-05-2024.

[65] .Net. ¿Qué es el çódigo administrado¿ https://learn.microsoft.com/es-es/dotnet/standard/managed-code, 2024. Accedido el 12-06-2024.

[66] Organización Mundial de la Salud. Diabetes. https://www.who.int/es/news-room/fact-sheets/detail/diabetes, 2023. Accedido el 18-02-2024.

[67] Overleaf. Overleaf Official Logos. https://es.overleaf.com/for/partners/logos. Accedido el 17-04-2024.

[68] Pokefanáticos. Pokémon GO presenta AR+, modo mejorado de Realidad Aumentada para dispositivos Apple. https://pokefanaticos.com/web/noticias/1219/pokemon-go-presenta-ar-modo-mejorado-de-realidad-aumentada-para-dispositivos-apple/, 2017. Accedido el 20-02-2024.

[69] Profesional review. Oculus Quest 2 Review. https://www.profesionalreview.com/2020/12/31/oculus-quest-2-review/#:~:text=Oculus%20Quest%202%20monta%20el,similar%20a%20un%20Snapdragon%20865. Accedido el 07-06-2024.

[70] Profesional review. Tipos gafas realidad virtual y cuáles son sus diferencias. https://www.profesionalreview.com/2023/03/04/tipos-gafas-realidad-virtual/. Accedido el 04-06-2024.

[71] Samsung. Gear VR. https://www.samsung.com/es/support/model/SM-R322NZWAPHE/. Accedido el 16-06-2024.

[72] Sonix. Cómo transcribir una reunión de Zoom. https://sonix.ai/es/how-to-transcribe-a-zoom-meeting. Accedido el 17-04-2024.

[73] Specification, O. A. Omg unified modeling language (omg uml), superstructure, v2. 1.2. *Object Management Group 70* (2007), 16. Accedido el 30-03-2024.

[74] The British Medical Journal. Global burden of type 2 diabetes in adolescents and young adults, 1990-2019: systematic analysis of the Global Burden of Disease Study 2019. https://www.bmj.com/content/379/bmj-2022-072385, 2022. Accedido el 18-02-2024.

[75] Tienda Lenovo. Lenovo Legion 5 15ITH6H - 82JH00LFSP. https://www.tiendalenovo.es/lenovo-legion-5-15ith6h-82jh00lfsp, 2024. Accedido el 24-03-2024.

[76] Trello. Trello facilita a los equipos la gestión de proyectos y tareas. https://trello.com/es/tour. Accedido el 15-04-2024.

[77] uc3m. OVERLEAF - EDITOR ONLINE LATEX. https://www.uc3m.es/sdic/servicios/overleaf. Accedido el 17-04-2024.

[78] Udemy. What is the Unity Game Engine? https://blog.udemy.com/unity-game-engine/. Accedido el 31-03-2024.

[79] Unity. Cada viaje creativo comienza con Unity Hub. https://unity.com/es/unity-hub. Accedido el 15-04-2024.

[80] Unity. Creación de scripts en Unity para programadores expertos. https://unity.com/es/how-to/programming-unity#:~:text=%C2%BFTienes%20experiencia%20en%20C%2B%2B%20y%20est%C3%A1s%20empezando%20a%20utilizar%20Unity%3F&text=C%23%20es%20m%C3%A1s%20f%C3%A1cil%20de,las%20fugas%20de%20memoria%2C%20etc. Accedido el 15-04-2024.

[81] Unity. XR Device Simulator. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/xr-device-simulator.html. Accedido el 08-06-2024.

[82] Unity. XR Interaction Toolkit. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html. Accedido el 08-06-2024.

[83] Unity. PLANES Y PRECIOS, Comienza a crear con Unity. https://unity.com/es/pricing#plans-individualsand-teams, 2024. Accedido el 24-03-2024.

[84] Unity. PLANES Y PRECIOS, Comienza a crear con Unity. https://unity.com/es/pricing#plans-student-and-hobbyist, 2024. Accedido el 24-03-2024.

[85] Unity Asset Store. Athletics Track. https://assetstore.unity.com/packages/3d/environments/roadways/athletics-track-74899, 2024. Accedido el 24-03-2024.

[86] Unity Documentation. Android environment setup. https://docs.unity3d.com/Manual/android-sdksetup.html. Accedido el 04-06-2024.

[87] Unity Documentation. Canvas. https://docs.unity3d.com/es/530/Manual/GameObjects.html. Accedido el 21-05-2024.

[88] Unity Documentation. Escenas. https://docs.unity3d.com/es/2018.4/Manual/CreatingScenes.html. Accedido el 21-05-2024.

[89] Unity Documentation. Flujo de trabajo de los Assets (Asset Workflow). https://docs.unity3d.com/es/530/Manual/AssetWorkflow.html#:~:text=Un%20asset%20es%20una%20representaci%C3%B3n,de%20archivos%20que%20Unity%20soporta. Accedido el 21-05-2024.

[90] Unity Documentation. GameObjects. https://docs.unity3d.com/es/530/Manual/GameObjects.html. Accedido el 21-05-2024.

[91] Unity Documentation. Gestión de complementos XR. https://docs.unity3d.com/Manual/com.unity.xr.management.html. Accedido el 08-06-2024.

[92] Unity Documentation. MonoBehaviour. https://docs.unity3d.com/Manual/class-MonoBehaviour.html. Accedido el 26-05-2024.

[93] Unity Documentation. Prefabs. https://docs.unity3d.com/es/2018.4/Manual/Prefabs.html. Accedido el 21-05-2024.

[94] Unity Documentation. Collider.isTrigger. https://docs.unity3d.com/ScriptReference/Collider-isTrigger.html, 2024. Accedido el 29-03-2024.

[95] Universidad de Valladolid. Guía Docente del Trabajo de Fin de Grado (mención Ingeniería de Software). https://apps.stic.uva.es/guias_docentes/uploads/2023/545/46976/1/Documento.pdf, 2023. Accedido el 10-03-2024.

[96] University of California, San Francisco. Tipos de insulina. https://dtc.ucsf.edu/es/tipos-de-diabetes/diabetes-tipo-2/tratamiento-de-la-diabetes-tipo-2/medicamentos-y-terapias-2/prescripcion-de-insulina-para-diabetes-tipo-2/tipos-de-insulina/. Accedido el 20-02-2024.

[97] Versus. Meta Quest 2 vs Oculus Quest. https://versus.com/es/meta-quest-2-vs-oculus-quest. Accedido el 07-06-2024.

[98] Villarejo, L. Realidad virtual aplicada a la formación en salud. https://blogs.uoc.edu/cienciasdelasalud/es/realidad-virtual-aplicada-a-la-formacion-en-salud/, 2019. Accedido el 26-02-2024.

[99] Visual Paradigm. What is Deployment Diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/. Accedido el 13-05-2024.

[100] Voiped. Microsoft Teams Operator Connect: Ventajas. https://www.voiped.com/es/blog/microsoft-teams-operator-connect-ventajas/. Accedido el 17-04-2024.

[101] Wikipedia. Archivo:GitLab logo.svg. https://es.wikipedia.org/wiki/Archivo:GitLab_logo.svg. Accedido el 17-04-2024.

[102] Wikipedia. Archivo:Trello-logo-blue.svg. https://es.m.wikipedia.org/wiki/Archivo:Trello-logo-blue.svg. Accedido el 17-04-2024.

[103] Wikipedia. Microsoft Outlook. https://es.wikipedia.org/wiki/Microsoft_Outlook. Accedido el 17-04-2024.

[104] Wikipedia. Oculus Quest. https://en.wikipedia.org/wiki/Oculus_Quest. Accedido el 07-06-2024.

[105] Wikipedia. Unity Technologies. https://es.wikipedia.org/wiki/Unity_Technologies. Accedido el 18-04-2024.

[106] y Mike Cotterell, B. H. *Software Project Management, 5ª edición*. McGraw-Hill Higher Education, 2009. Accedido el 02-03-2024.

[107] Yair Carreno. Conceptos sobre Inversión de dependencia, Inversión de control (IoC) e Inyección de dependencias (DI). https://yaircarreno.medium.com/conceptos-sobre-inversi%C3%B3n-de-dependencia-inversi%C3%B3n-de-control-ioc-e-inyecci%C3%B3n-de-dependencias-6e95c004951a, 2022. Accedido el 13-05-2024.

[108] Zoom. Zoom Workplace. https://zoom.us/es. Accedido el 17-04-2024.

Apéndice A

Manual de usuario

Este manual indica información relevante para el usuario sobre la utilización del videojuego, como la navegación por las diferentes escenas o como moverse durante la carrera. SugarRace está compuesto por diferentes interfaces de usuario que se explicarán a continuación:

- **Menú inicial:** Es la primera interfaz que aparece al iniciar el videojuego (Figura A.1). En ella el usuario podrá familiarizarse con el entorno ya que podrá rotar la cámara en todas las direcciones. El principal objetivo de esta pantalla es que el usuario seleccione uno de los niveles disponibles mediante los controladores de realidad virtual.
- **Menú de toma de decisiones:** Una vez seleccionado el nivel, aparecerá esta interfaz donde el usuario deberá seleccionar cuántas raciones de alimento quiere ingerir y de qué tipo, cuántas dosis de insulina se quiere aplicar y cuánto tiempo va a esperar antes de comenzar a hacer ejercicio físico. Una vez seleccionado se debe dar al botón de jugar. Esta interfaz aparece en la Figura A.2
- **Juego:** Durante la carrera el usuario únicamente podrá moverse horizontalmente y no tendrá la capacidad de rotar la cámara. Para poder moverse de vía se utilizará el Joystick del controlador izquierdo. Durante el juego aparecerán diferentes objetos que podrán ser recogidos si se viera conveniente.

  En la interfaz también se indica arriba a la izquierda el número de puntos del usuario que se irá modificando a lo largo de la carrera dependiendo de los objetos que se recojan y del nivel de glucosa del usuario al llegar a un punto de control. Arriba a la derecha se muestra el tiempo restante del nivel y justo debajo el nivel de glucosa, representada tanto con valor numérico como con una barra. En la Figura A.3 aparece dicha interfaz.
- **Pantalla de fin de nivel:** Al finalizar el nivel automáticamente aparecerá esta interfaz que muestra información relevante del nivel. Si se ha superado o, en el caso de que no, el motivo por el cual se ha perdido el nivel. El tiempo en rango que representa el tiempo que el usuario se ha mantenido en un rango saludable durante el juego, expresado en

porcentaje. El número de puntos obtenido y una frase educativa sobre la diabetes tal y como se muestra en la Figura A.4

Figura A.1: Interfaz de usuario del menú inicial.

Figura A.2: Interfaz de usuario del menú de decisiones iniciales.

Figura A.3: Interfaz de usuario de la carrera.

Figura A.4: Interfaz de usuario del la pantalla de fin de nivel.

Apéndice B

Manual de despliegue y ejecución

B.1. Despliegue del proyecto en Unity

Este manual proporciona información sobre versiones e instrucciones para poder desplegar el proyecto en Unity a partir del repositorio de GitLab.

B.1.1. Requisitos del sistema

- Versión Windows 10.
- Versión de Unity Hub 3.7.0 o superior.
- Versión de Unity Editor 2022.3.13f1 o superior.
- Versión de Visual Studio 17.8.0 o superior.

B.1.2. Proceso de despliegue

1. **Clonar el repositorio de GitLab**, que se puede realizar mediante el uso de Visual Studio o de cualquier otro IDE, abriendo una terminal y ejecutando el comando:

   `git clone https://gitlab.com/HP-SCDS/Observatorio/2023-2024/diaberse/uva-diaberse.git`

2. **Abrir el proyecto en Unity Hub**, seleccionando la opción *Projects* y se debe pulsar el botón "Add" donde habrá que buscar el proyecto clonado y seleccionarlo.

3. **Configuración del dispositivo de realidad virtual:**

   - Seleccionar dentro del Editor de Unity la pestaña *Edit =>Project Settings =>XR Plug-in Management* y desplegar esta opción para ver los *Plug-in Providers* que ya están configurados.

- Seleccionar la opción *OpenXR* y en la sección *Enabled Interaction Profiles* añadir las gafas de realidad virtual mediante el símbolo "+" y buscando el dispositivo.
- *OpenXR* debe quedar en la parte superior de la lista ya que al ejecutarse la aplicación seguirá ese orden.

Para usuarios que no dispongan de gafas de realidad virtual, es posible generar un ejecutable del videojuego donde se utilizará un simulador de realidad virtual.

4. **Creación del ejecutable** desde la pestaña *File =>Build Settings =>Build And Run.* Una vez finalizado la creación del ejecutable se abrirá en las gafas de realidad virtual que estemos utilizando.