UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Juego de realidad virtual sobre recogida de setas: UBUSetasVR 1.0

Presentado por Rodrigo Jurado Pajares
en Universidad de Burgos — 25 de junio de 2018
Tutor: D. José Francisco Díez Pastor y D. Raúl Marticorena Sánchez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. José Francisco Díez Pastor y D. Raúl Marticorena Sánchez, profesores del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos.

Exponen:

Que el alumno D. Rodrigo Jurado Pajares, con DNI 71298346W, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado "Juego de realidad virtual sobre recogida de setas".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 25 de junio de 2018

Vº. Bº. del Tutor:

D. José Francisco Díez Pastor

Vº. Bº. del Tutor:

D. Raúl Marticorena Sánchez

Resumen

Este proyecto trata sobre la realización de un videojuego de Realidad Virtual para dispositivos móviles Android de alta gama utilizando la plataforma Daydream de Google para entornos de Realidad Virtual. El juego consiste en la recogida de diferentes especies reales de setas en entornos de bosques ficticios, haciendo uso de una base de datos con información acerca de las setas. Esta base de datos contendrá aspectos como la descripción, el género o las propiedades comestibles de las setas, y con esta información el jugador deberá catalogar las setas según ciertos parámetros y sumará o restará puntos en base a si ha acertado o ha fallado con sus decisiones. El juego tiene como objetivo ser una herramienta complementaria de aprendizaje sobre micología. Para su realización se ha utilizado la herramienta de creación de videojuegos Unity3D.

Descriptores

Realidad Virtual, Daydream, Unity3D, móvil, Android, videojuego, setas

Abstract

This project is about the process of making a Virtual Reality videogame for high-end Android mobile platforms using the Daydream platform from Google. The game consist of picking real mushrooms species in different fictitious forests using a database with information like their description, genre or edibility. The player will use this information to catalogue mushrooms following certain parameters and will increase or decrease score based on the success of the decisions made. The game objective is to be a complementary tool to enhance learning about mycology. The game is made with Unity3D.

Keywords

Virtual Reality, Daydream, Unity3D, mobile, Android, videogame, mushrooms

Índice general

Índice de figuras

Índice de tablas

Introducción

La industria del videojuego se ha convertido en el motor principal del entretenimiento global. En 2017 ha sido capaz de generar 116.000 millones de dólares a nivel global y tener 2.200 millones de jugadores de todas las edades, donde las mujeres representan el 47 % de la cifra. Solo en videojuegos para dispositivos móviles ha generado 50.000 millones de dolares y ha reunido a 2.000 millones de jugadores en torno a ellos, siendo el segmento principal de mercado de esta industria, y los pronósticos para los próximos años es que estas cifras sigan incrementándose, según indica el *Libro blanco del Desarrollo Español de Videojuegos* [5] en la figura 1.1.

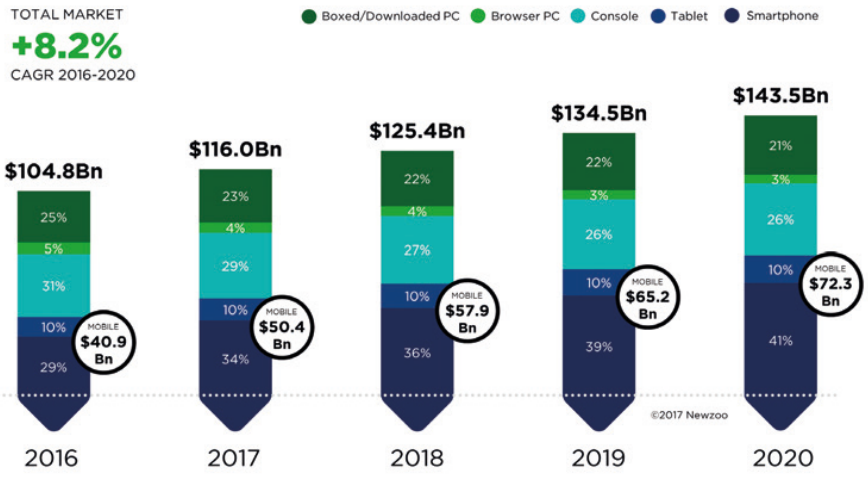

Figura 1.1: Evolución prevista del mercado mundial de videojuegos

Por otro lado, la Realidad Virtual (RV) está situado como uno de los segmentos del mercado con mayor crecimiento en 2017 con un espectacular 259 % respecto a 2016, cifra destacable en comparación al crecimiento de otros

segmentos en ese mismo año, tal y como se puede ver en la figura 1.2, aunque todavía no genera un volumen de negocio relevante y la mayor parte de los ingresos generados, 1,8 billones de dolares en 2016, son origen de la venta de hardware. Los videojuegos se llevan aún así el 44 % de los ingresos generados por venta de software y servicios en RV, como se puede apreciar en la figura 1.3.

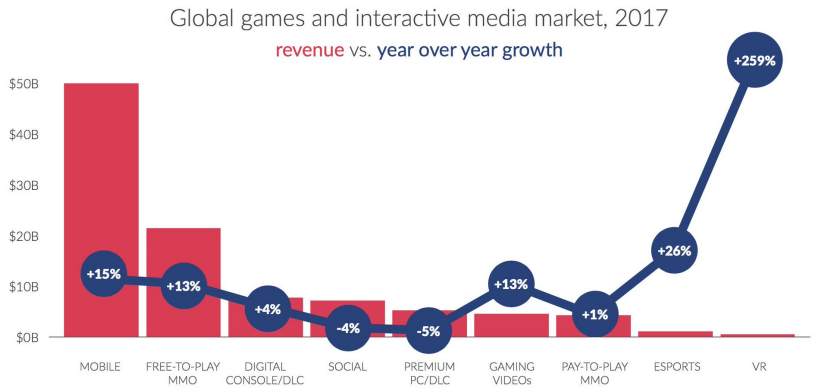

Figura 1.2: Mercado mundial del videojuego en 2017 por modelo de negocio [48].

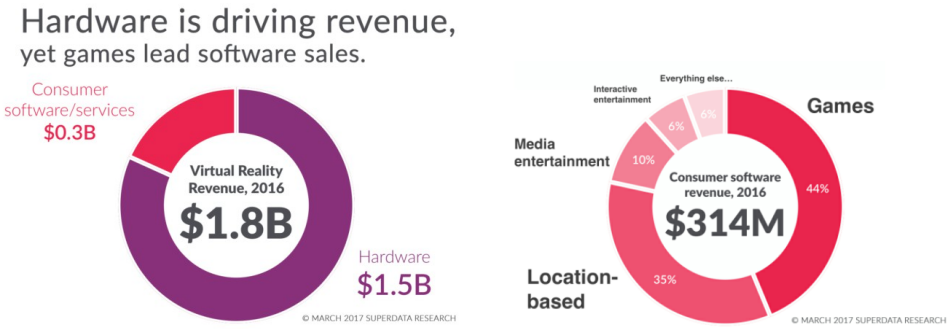

Figura 1.3: Ingresos generados por la Realidad Virtual en 2016 e ingresos generados por venta de software de RV según segmentos [48].

Con estas cifras se puede ver que tanto el mercado de los videojuegos como los sectores de dispositivos móviles y de RV presentan grandes números, oportunidades y crecimiento. Estas son algunas de las principales motivaciones a la hora de realizar este proyecto para poder presentar la capacidad actual que tienen los videojuegos de RV en dispositivos móviles, tanto a nivel de herramientas como a nivel de resultados.

1.1. Estructura de la memoria

La memoria tiene la siguiente estructura:

- **Introducción:** breve descripción del problema a resolver y la solución propuesta. Estructura de la memoria y listado de materiales adjuntos.
- **Objetivos del proyecto:** exposición de los objetivos que persigue el proyecto.
- **Conceptos teóricos:** breve explicación de los conceptos teóricos clave para la comprensión de la solución propuesta.
- **Técnicas y herramientas:** listado de técnicas metodológicas y herramientas utilizadas para la gestión y desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** exposición de aspectos destacables que tuvieron lugar durante la realización del proyecto.
- **Trabajos relacionados:** estado del arte en el campo de la Realidad Virtual y su uso en dispositivos móviles y proyectos relacionados.
- **Conclusiones y líneas de trabajo futuras:** conclusiones obtenidas tras la realización del proyecto y posibilidades de mejora o expansión de la solución aportada.

Anexos aportados junto a la memoria:

- **Plan del proyecto software:** planificación temporal y estudio de viabilidad del proyecto.
- **Especificación de requisitos del software:** se describe la fase de análisis; los objetivos generales, el catálogo de requisitos del sistema y la especificación de requisitos funcionales y no funcionales.
- **Especificación de diseño:** se describe la fase de diseño; el ámbito del software, el diseño de datos, el diseño procedimental y el diseño arquitectónico.
- **Manual del programador:** recoge los aspectos más relevantes relacionados con el código fuente (estructura, compilación, instalación, ejecución, pruebas, etc.).
- **Manual de usuario:** guía de uso de la aplicación para el usuario.

1.2. Materiales adjuntos

Materiales adjuntos a la memoria:

- Aplicación para Android del videojuego.
- Vídeos de demostración de la aplicación.
- Modelos digitales 3D utilizados y vídeos de alta calidad para visualizar los modelos.
- Hoja de cálculo con los resultados del análisis de calidad del código.

Además, los siguientes materiales están accesibles en internet:

- Aplicación en Google Play[1] [36].
- Repositorio del proyecto [35].

[1] La aplicación se encuentra subida de forma privada como Alfa Cerrada y es necesario tener una cuenta autorizada por el propietario para poder acceder a la ficha de la aplicación. Más información en el apartado 5.7.

Objetivos del proyecto

2.1. Objetivos generales

- Desarrollar un videojuego para la plataforma móvil Android que haga uso de Realidad Virtual.
- Proporcionar herramientas para facilitar el crecimiento del videojuego en un futuro.
- Utilizar el concepto de *Gamificación* o *Ludificación* para enseñar acerca de diferentes especies de setas y su entorno.
- Facilitar la entrada del público general en el mundo de la Realidad Virtual.

2.2. Objetivos técnicos

- Desarrollar una aplicación Android con soporte para Android 7.0 'Nougat' (API 24) y superiores.
- Utilizar Daydream como plataforma de Realidad Virtual.
- Utilizar Daydream View como visor de Realidad Virtual.
- Utilizar Git como sistema de control de versiones junto a la plataforma GitHub.
- Aplicar la metodología ágil Scrum en el desarrollo del software.
- Utilizar ZenHub como herramienta de gestión de proyectos.
- Distribuir la aplicación resultante en la plataforma Google Play.

2.3. Objetivos personales

- Poner en práctica mis conocimientos sobre Realidad Virtual y desarrollo de videojuegos.
- Mejorar mi experiencia en el desarrollo de videojuegos con Unity3D para proyectos futuros.
- Profundizar en el mundo de la Realidad Virtual en dispositivos móviles.
- Explorar acerca de la accesibilidad de nuevas tecnologías.
- Utilizar y perfeccionar el uso de metodologías y herramientas de gran utilidad en el mercado laboral.

Conceptos teóricos

3.1. Motor Gráfico

Un motor gráfico o motor de videojuego [67] es un software que aglutina una serie de rutinas de programación con el fin de permitir la edición, diseño y presentación gráfica de un videojuego. Los motores gráficos integran diferentes componentes, siendo el más importante el *motor de renderizado*, que se encarga de generar los gráficos en 2D o 3D, y suelen incorporar sistemas de físicas, de animación o de inteligencia artificial entre otros. A la hora de desarrollar un videojuego, una de las primeras elecciones es el motor gráfico a utilizar y marcará el resto del desarrollo así como el producto final obtenido. En este proyecto, el motor gráfico utilizado es Unity3D.

3.2. Fotogramas por segundo

Los fotogramas por segundo [38, 66], *frames per second*, o FPS para abreviar[2], es una de las medidas de rendimiento más utilizada en el mundo de los videojuegos. Como su nombre indica, refleja los fotogramas que se muestran por pantalla en un segundo, o lo que es lo mismo, la tasa de refresco de las imágenes que se muestran por pantalla. Esta tasa marca el tiempo que transcurre entre que se ejecuta la lógica del juego, se hacen los cálculos matemáticos gráficos necesarios y se muestra por pantalla el resultado, por lo que es un buen indicador del rendimiento del juego en líneas generales. Actualmente, en la mayoría de desarrollos de videojuegos profesional se persigue el objetivo de 60 FPS estables [28], en comparación a los 24 FPS que se utilizaban tradicionalmente en producciones cinematográficas [22]. Para Realidad Virtual es aconsejable, y a veces incluso necesaria, una tasa de refresco muy superior, como se explicará más adelante.

[2]La abreviatura FPS también puede referirse a *First Person Shooter*, o videojuego de disparos en primera persona, aunque en esta memoria no se utilizará con ese significado.

3.3. Unity3D

Unity3D [53] es el motor gráfico utilizado para este proyecto. Es un motor basado en C++ nativo que proporciona un amplio abanico de sistemas que facilitan la creación y desarrollo de un videojuego. Cuenta con una gran comunidad de desarrolladores muy extendida globalmente que ha generado una gran cantidad de documentación acerca del motor, facilitando así su aprendizaje y uso. El motor es gratuito para uso personal o no comercial, teniendo tramos de licencias para uso profesional con soporte y características extendidas [61].

C# en Unity3D

Unity3D tiene soporte para las librerías de clases de .NET [55, 54], tanto para la versión 2.0 como para la 4.6, lo que permite usar el lenguaje de programación orientado a objetos C# de forma nativa. Esto hace que C# sea el lenguaje de programación principal de Unity3D, y el hecho de que la empresa desarrolladora se haya unido a la *.NET Foundation* [50] hace que gane aun más importancia como lenguaje principal. Aun así, Unity3D permite el uso de otros lenguajes como JavaScript, aunque por detrás adaptará el código, ya que este se compila y ejecuta con el *framework* de .NET, por lo que dependiendo de lo actualizadas que estén las librerías de compatibilidad estos otros lenguajes pueden no dar tanta versatilidad y rendimiento en la multitud de plataformas soportadas por Unity3D [56].

Escenas

Una escena [59] es un concepto propio de Unity3D y uno de los elementos principales de trabajo en la plataforma. Es el elemento mínimo para conseguir un producto mínimo viable ejecutable, ya que contiene todo el entorno, los objetos y los menús del juego, permitiendo que este se vaya montando "a piezas" a partir de como mínimo una escena. Se suele asociar la idea de escena con la de un nivel del juego, aunque no siempre tiene por qué ser así.

GameObjects

El concepto de GameObject [58] es el más importante en Unity3D. Todos los objetos presentes en el juego, ya sean personajes, luces, terrenos, efectos, cámaras, etc. son GameObjects. Es el elemento mínimo de trabajo que puede aparecer en una escena. Un GameObject u objeto en sí no tiene ninguna propiedad, es simplemente un contenedor vacío al que se le van añadiendo componentes, y es gracias a ellos que cada objeto adquiere propiedades diferentes a otros objetos.

Componentes

Un componente [57] es una pequeña pieza funcional que aporta una funcionalidad concreta a un GameObject u objeto. Un ejemplo de componente sería un Transform, que aporta una funcionalidad espacial a un objeto para que este pueda ser representado en base a unas coordenadas en una escena, o un Rigidbody, que aporta la funcionalidad de propiedades físicas a un objeto, permitiendo así que este sea afectado por una simulación (aproximada) de las leyes físicas.

ScriptableObjects

Un ScriptableObject [60] es un tipo especial de clase en Unity destinada al almacenamiento de datos principalmente. Su función es mantener datos compartidos de forma independiente y única a cualquier tipo de instancia, es decir, mantiene los datos guardados incluso cuando la aplicación no se encuentra en ejecución y no importa cuantos objetos utilicen esos datos, solo existirá una única copia en memoria, tal y como se puede ver en la figura 3.4. Este comportamiento de existencia única incluso con múltiples usos no es el funcionamiento habitual en el resto de objetos de Unity, los cuales van ocupando memoria conforme se van utilizando, independientemente de si los datos usados son los mismos o no.

Por ejemplo, si existe una lista de elementos que ocupa 10MB y se utiliza cuatro veces con objetos normales, este uso ocupará 40MB en memoria en total. Si esa misma lista fuese implementada mediante un ScriptableObject, podría usarse 100 veces y el tamaño total en memoria seguiría siendo de 10MB.

3.4. Realidad Virtual

Se conoce por Realidad Virtual [24] al escenario generado por ordenador que simula una experiencia realista. Este escenario permite que, mediante el estímulo de los sentidos del usuario, se consiga cierto grado de inmersión en el mundo artificial generado. La libertad que posea el usuario dentro de dicho mundo artificial o virtual dependerá de varios factores, que se comentarán a continuación.

Grados de libertad

Los grados de libertad [25, 73], o *Degrees of Freedom* (DOF) en inglés, son las formas en las que un objeto se puede mover dentro de un espacio. Hay un total de 6 grados de libertad en un espacio tridimensional, los cuales se pueden dividir en dos categorías de movimientos, los movimientos de rotación y los movimientos de traslación. Cada una de las categorías tiene 3 grados

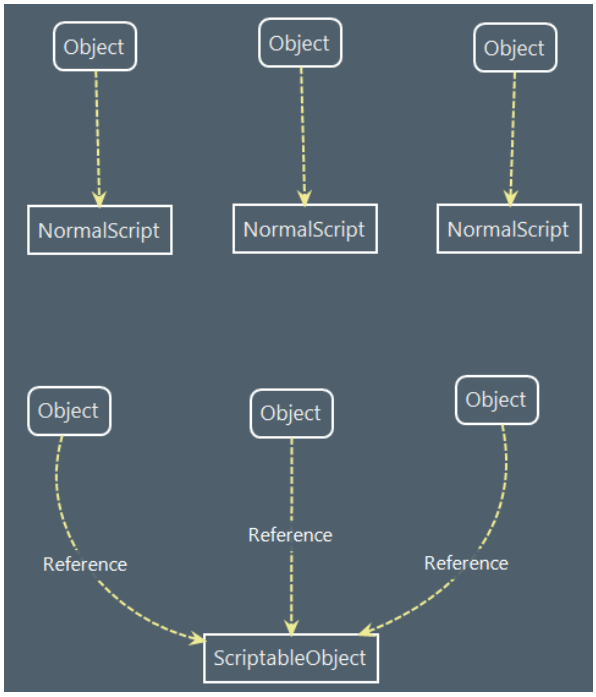

Figura 3.4: Funcionamiento de un *ScriptableObject* en comparación a un *GameObject* [63].

de libertad, y para conseguir una buena inmersión en un mundo virtual, se requiere alcanzar los 6 grados de libertad.

Movimientos de rotación - 3 grados de libertad

Hay tres movimientos de rotación [71] que, llamados según la nomenclatura aeronáutica, son: guiñada, cabeceo y alabeo (yaw, pitch, roll en inglés). Estos movimientos se producen cuando hay movimiento alrededor de uno de los tres ejes de un sistema de referencia tridimensional. En Realidad Virtual, estos tres tipos de movimientos proporcionan el seguimiento de rotación del usuario y son bastante fáciles de seguir con un dispositivo que cuente con acelerómetro, giroscopio y magnetómetro, componentes presentes actualmente en casi la totalidad de dispositivos móviles por ejemplo. Como se ve en la figura 3.5.

Movimientos de rotación y traslación - 6 grados de libertad

Si a los movimientos de rotación se les añade los movimientos de traslación, se alcanzan los 6 grados de libertad [70]. Existen tres movimientos de traslación: elevación, movimiento lateral y movimiento frontal (elevation, strafe, surge en inglés). Estos movimientos se producen cuando hay movimiento a lo largo de uno de los tres ejes de un sistema de referencia tridimensional. En

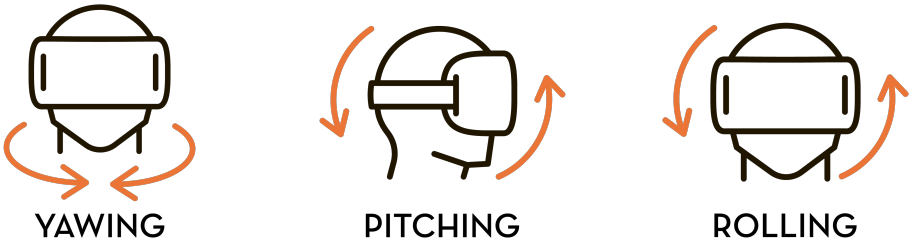

Figura 3.5: Representación gráfica de los movimientos de rotación. De izq. a der.: guiñada, cabeceo y alabeo [47].

Realidad Virtual, estos tres tipos de movimientos proporcionan el seguimiento posicional del usuario y son difíciles de seguir sin ayuda de un dispositivo externo que ayude a posicionar al usuario en el espacio real.

Figura 3.6: Representación gráfica de los movimientos de traslación. De izq. a der.: elevación, mov. lateral y mov. frontal [47].

Campo de visión

El campo de visión [6, 23], o *field of view* (FOV) en inglés, es la extensión del mundo observable en un momento dado. El ojo humano tiene un campo de visión de 170º-175º, siendo 60º el campo de visión en dirección a la nariz y 110º el campo de visión hacia los laterales de la cabeza. Cuando se combinan los campos de visión de los dos ojos, se obtiene el llamado campo visual binocular, una región que abarca unos 114º en la cual se puede apreciar profundidad de campo y por tanto percibir objetos en tres dimensiones. En la figura 3.7 se muestra una representación de los campos visuales comentados y se ve el campo visual de un conejo con el de un humano a modo de comparativa. Los primeros cascos de realidad virtual ofrecían un campo de visión binocular simulado de 90º usando lentes de diferentes grosores y curvaturas, así como ajustes en la distancia interpupilar (distancia entre el centro de las pupilas) para que la experiencia de Realidad Virtual se adaptase a cualquier persona.

Actualmente, los cascos del mercado ofrecen un campo de visión de 110º y hay proyectos de cascos con campos de visión cercanos a los 200º.

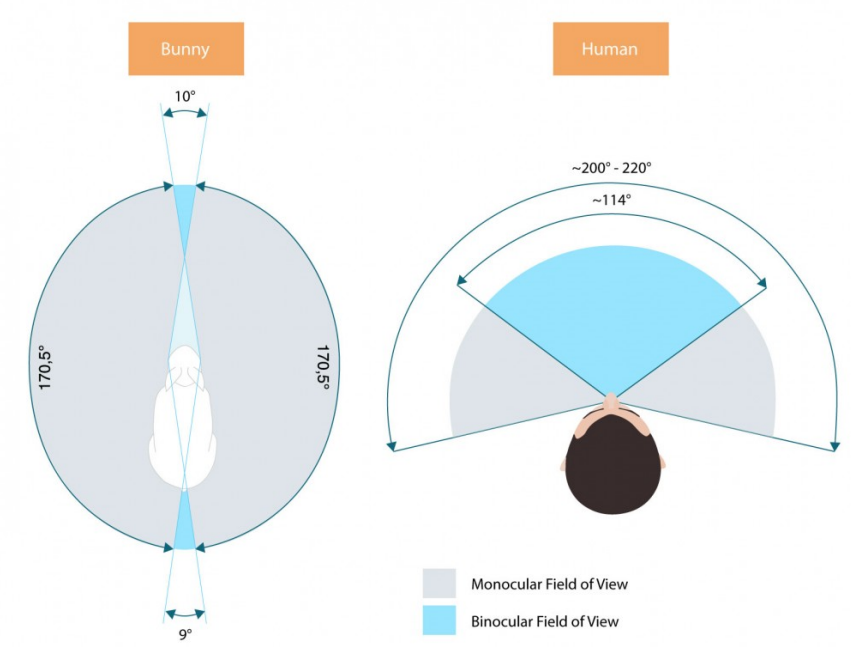

Figura 3.7: Comparación visual del campo de visión de un conejo con el de un humano [23].

Interacción

La interacción con objetos virtuales se puede producir de diferentes formas, siendo la forma más común el uso de un controlador para ello, aunque este controlador puede tener diferentes características dependiendo del modelo y del fabricante. Por ejemplo, el HTC-Vive [68] usa dos controladores, uno para simular cada mano, y cada controlador cuenta con sistema de posicionamiento y localización propio, mientras que el Oculus Rift [69] usa un controlador que solo proporciona un acelerómetro, un giroscopio y un magnetómetro. Existen actualmente una corriente de nuevos periféricos destinados a la interacción con el mundo virtual, aunque la mayoría de ellos están aun en una fase temprana de desarrollo. Como ejemplos destacados, mencionar la plataforma de movimiento Omni [72], con la cual se puede trasladar el movimiento real del usuario al mundo virtual y que este movimiento no se vea limitado por el espacio físico real, o los GloveOne [49], unos guantes que permiten simular los movimientos de las manos del usuario en el mundo virtual y recrear la sensación del tacto al producir estímulos sensoriales en las manos en base a eventos del mundo virtual. En la figura 3.8 se pueden ver los sistemas de interacción

mencionados, teniendo así una comparativa visual de lo diferente que pueden ser los periféricos destinados a RV.

Figura 3.8: Diferentes periféricos de interacción en Realidad Virtual. De izq. a der.: HTC Vive (casco + controladores + estaciones de seguimiento), controladores Oculus Rift Touch, plataforma Virtuix Omni y guantes GloveOne

Rendimiento

En un videojuego el rendimiento es algo muy importante para la experiencia de usuario, y en Realidad Virtual hay que destacar sobretodo la gran importancia del rendimiento gráfico, que tiene mucho impacto sobre las tasas de refresco de pantalla y marcará en gran medida el rendimiento del juego apreciado por el usuario. En un videojuego se tiene por objetivo alcanzar los 60 FPS estables, en Realidad Virtual la cifra alcanza, e incluso supera, los 90 FPS estables, aunque en dispositivos móviles la cifra objetivo suele estar en torno a los los 60-70 fps estables debido a la gran diversidad de dispositivos móviles presentes en el mercado, cada uno con especificaciones técnicas muy dispares y diversas [44]. Remarcar la importancia de que sean FPS estables, ya que si se producen picos de rendimiento o los FPS son muy bajos el usuario experimentará con casi total seguridad mareos y nauseas, echando por tierra toda la experiencia. Tener que cuidar el apartado gráfico añade un extra de complicación y trabajo en Realidad Virtual, ya que hay que mantener el balance entre rapidez de procesamiento de cálculos y la calidad gráfica. Esto supone aun más reto cuando la Realidad Virtual está destinada a ejecutarse en un terminal móvil, ya que estos dispositivos suelen contar con muchas

menos prestaciones que un terminal de sobremesa, sobretodo en rendimiento gráfico.

3.5. Daydream

Daydream [12, 13] es una plataforma para Realidad Virtual en dispositivos móviles de gama alta creada por Google, una versión mejorada del sistema Cardboard [11] que tenía por objetivo dispositivos móviles de gama baja-media. Con Daydream, Google crea una serie de pautas y requisitos que los fabricantes y desarrolladores tienen que seguir si quieren que sus productos sean compatibles con la plataforma. Por ejemplo, los requisitos para publicar una aplicación compatible [14], o *Daydream-ready*, son bastante exigentes y contemplan aspectos visuales, de rendimiento o de experiencia de usuario entre otros. Daydream utiliza GoogleVR [18], un SDK[3] propio que proporciona las herramientas necesarias para implementar Realidad Virtual en dispositivos móviles de diferentes gamas y que tiene soporte en varios entornos de desarrollo.

Figura 3.9: Logotipo de la plataforma de Realidad Virtual Daydream.

Daydream View

Daydream View [16] es el nombre del paquete de hardware (casco + mando) que Google ha desarrollado para la Realidad Virtual. El casco es simplemente una carcasa vacía, de buena calidad, con lentes y con hueco para alojar un móvil en el extremo permitiendo una proyección binocular de la simulación. No cuenta con ninguna capacidad de procesamiento, siendo exclusivamente el móvil el que se encargue de la simulación y de la detección de movimientos de la cabeza gracias a los sensores propios del móvil. Esto hace que el casco solo tenga 3 grados de libertad, limitando la experiencia y la interacción, pero gracias al mando se pueden llegar a los 6 grados de libertad simulados [15]. El mando es un pequeño dispositivo inalámbrico que se conecta al móvil mediante Bluetooth y tiene sus propios sensores de movimiento, permitiendo así que se pueda interactuar con el mando libremente en la Realidad Virtual sin depender de los movimientos del casco. Este mando cuenta con un par de botones

[3] *Software Development Kit*, o kit de desarrollo de software

frontales, un par de botones laterales y una superficie táctil que puede actuar como botón también, lo que permite configurar unos cuantos controles usando combinaciones de teclas, aunque en ocasiones se puede echar en falta tener más botones para poder jugar más con las configuraciones de los controles.

Figura 3.10: Casco + mando que forman el Daydream View (izq) y esquema de botones del mando (der).

Técnicas y herramientas

4.1. Metodologías

Metodologías de trabajo contempladas y/o aplicadas en el proyecto

Scrum

Scrum [41] es un método de trabajo ágil para el desarrollo de software que aplica trabajo dividido en tareas, iterativo e incremental en el que se trabaja en unidades de tiempo definidas llamadas iteraciones o *sprints*, con una revisión del trabajo realizado al finalizar cada iteración. Este método de trabajo encaja perfectamente en el desarrollo del proyecto, ya que un videojuego se puede dividir perfectamente en tareas concretas relacionadas con funcionalidades que tras su desarrollo se pueden incorporar a un producto de prueba, haciendo que el juego vaya creciendo hasta llegar al producto final con todas las funcionalidades planificadas.

Gitflow

Gitflow [4] es una metodología de trabajo con Git para organizar las diferentes ramas existentes en el control de versiones en base a un tiempo de vida y a una funcionalidad asignada, que encaja perfectamente con una forma de trabajo iterativa como Scrum. Esta metodología define dos ramas principales con un tiempo de vida infinito, *master* y *develop*, que tendrán exclusivamente lo que consideraríamos versiones completas del software, la primera contendría versiones finales y la segunda versiones de desarrollo. Para cada nueva funcionalidad que se quiera añadir al software, se habilitarían nuevas ramas desde *develop* cuyo tiempo de vida sería el tiempo de desarrollo de la funcionalidad. Una vez terminada la funcionalidad, dicha rama se integra de nuevo en *develop*, incrementando así el contenido de la versión de desarrollo. Cuando se tenga una versión de desarrollo que cumpla los criterios de versión final, se in-

tegraría la rama de desarrollo en la rama *master*, completando así el desarrollo de una nueva versión final del producto.

Esta metodología, como se ha podido ver, se basa en gestionar de forma ordenada la integración continua de nuevas funcionalidades en el producto, obteniendo además productos finales o de prueba. Por ello, se incorporó esta metodología al proyecto desde un principio, aunque con el tiempo se fueron viendo problemas durante el desarrollo, los cuales serán comentados en otro apartado 5.1

4.2. Patrones de diseño

Algunos de los patrones de diseño más destacados usados en el proyecto.

Managers

El patrón de diseño basado en *managers* o gestores, o como es conocido formalmente, patrón fachada [1], se ha utilizado en este proyecto para tener un elemento general de control de funcionalidades, algo muy utilizado en el mundo de los videojuegos. De esta forma se tiene un elemento general de control que permite tocar ajustes relativos al juego por ejemplo, o cambiar las propiedades de un nivel o de algún elemento en concreto. Esto permite tener más encapsuladas las funcionalidades generales del juego, que de otra forma estarían repartidas en varios elementos del nivel o del juego, haciendo más difícil su mantenimiento.

Singleton

El patrón Singleton [1] es otro de los patrones más usados en videojuegos y se encarga de asegurar la existencia única de una instancia de una clase concreta al mismo tiempo. Un ejemplo de aplicación en este proyecto serían los managers mencionados anteriormente, ya que solo es necesario tener uno de cada tipo al mismo tiempo en escena. Hay discusiones no obstante acerca de este patrón y de si es una buena práctica utilizarlo en entornos de producción, ya que tiene un comportamiento muy similar a las demonizadas "variables globales" y empeora la posibilidad de utilizar pruebas unitarias, si bien es cierto que con un uso medido y controlado son de gran utilidad y agilizan el prototipado y en proyectos de esta envergadura no suelen suponer mucho problema.

4.3. Herramienta de gestión del proyecto

Se planteó el uso de Trello [64] inicialmente como herramienta de gestión de proyecto, pero se descartó rápidamente por la escasez de opciones de in-

tegración con repositorios de código en su modelo gratuito. La herramienta elegida fue ZenHub [74], que permite integrar un completo sistema de gestión de proyectos acorde a la metodología Scrum y que cuenta entre otros cosas con un tablero de gestión de tareas y gráficas para el seguimiento del desarrollo de cada iteración.

4.4. Sistema de control de versiones

Aquí no hubo dudas acerca de qué herramienta utilizar: Git [3]. El control de versiones no está especialmente desarrollado (no de forma asequible o intuitiva) en los componentes propios de Unity3D, y pese a ello Git es uno de los mejores sistemas para hacer el control de versiones en Unity3D aun con sus limitaciones. Este sistema permite tener un control de versiones sobre el código del proyecto y, cambiando algunas opciones, sobre aspectos de Unity3D como por ejemplo las opciones de configuración de las escenas, aunque aun con esto no permite visualizar los cambios realizados en el código de las escenas en sí (qué objetos hay en la escena, los componentes de los objetos, etc.).

Alojamiento del repositorio

Para alojar el repositorio de Git de forma online se decidió usar GitHub [9], ya que además de contar con un *issue tracker*, permite integración con ZenHub, haciendo que GitHub pase a ser la herramienta central para la gestión del proyecto y de control de versiones.

Cliente de control de versiones

Como cliente de Git se ha utilizado SourceTree [45], un cliente creado por Atlassian que permite un control sencillo y visual de las diferentes ramas, *commits* y comandos de Git.

4.5. Motor gráfico

Hay varios motores gráficos disponibles en el mercado con licencias gratuitas o con modelos de regalías, aunque los más conocidos y extendidos por una cosa u otra son Unreal Engine [7], Lumberyard [2] y Unity3D [53]. Los dos primeros fueron descartados bastante rápido por la elevada curva de dificultad a la hora de aprender su funcionamiento, aunque no se puede negar la calidad de los resultados que proporcionan. El motor elegido ha sido Unity3D por contar con una gran comunidad, mucha documentación abierta y accesible, y sobretodo por tener ya conocimientos previos en dicho motor gráfico, facilitando así el desarrollo.

4.6. Entorno de programación

Al trabajar con el lenguaje de programación C#, lenguaje propietario de Microsoft, no ha habido mucha duda a la hora de elegir un IDE (Integrated Development Enviroment) con el que trabajar: Visual Studio [30]. Existían otras dos opciones que eran Rider [26] y MonoDevelop [32], pero ambas tenían algunos aspectos que no encajaban bien para este proyecto. Rider de Jetbrains era muy buena opción por lo potente que es el IDE y por estar ya acostumbrado al uso de otros productos de la compañía, pero todavía está en fase beta y no tiene licencia gratuita. Cuenta con licencia para estudiantes, pero se descartó igualmente por no ser viable a largo plazo. MonoDevelop es un IDE OpenSource bastante ligero y veloz que soporta .NET y hasta hace poco era el IDE oficial en Unity3D, pero la inminencia de discontinuidad del proyecto y la falta de integración con nuevas versiones de .NET hizo que VisualStudio cogiese el relevo, motivo por el cual se descartó también este entorno. Por lo tanto VisualStudio es el IDE elegido para el proyecto, ya que ahora es la opción oficial de Unity3D [34], es la herramienta por excelencia para .NET, y es la que mejor se integra con todo lo que puede ofrecer Unity3D.

4.7. Testing, integración continua y calidad del código

Testing automatizado

Automatizar los test en un producto software nunca es tarea sencilla, y es más complicado aun cuando hablamos de videojuegos [27] por la gran cantidad de variables y comportamientos del usuario a tener en cuenta en los test, así como los numerosos sistemas que suelen coexistir en un mismo juego. En Unity, la automatización de test es aun más complicada si cabe, ya que aunque cuenta con un sistema de testeo integrado, la interfaz de usuario, la usabilidad y la funcionalidad están bastante limitadas y dejan bastante que desear. Como ejemplo de los inconvenientes del sistema de testeo integrado de Unity, solo proporciona herramientas para test unitarios que no proporcionan ninguna abstracción o implementación básica para falsear las clases de conceptos propios de Unity, haciendo que se lancen numerosos falsos positivos en los test o, incluso, haciendo que ni compilen dichos test en algunas ocasiones. Esto obliga a tener que crear desde cero todo el sistema de test, haciendo inviable plantear tanto los test como un sistema automatizado en el periodo que abarca el desarrollo del proyecto.

Integración continua

Un proyecto con Integración Continua (CI) es un proyecto sano, en el sentido de que es fácilmente mantenible y, normalmente, fácilmente escalable.

Introducir CI en el proyecto era uno de mis objetivos personales antes de empezar el proyecto, pero lamentablemente me ha sido imposible introducirlo. Unity3D no contaba con una forma sencilla de hacer CI, no sin perder meses de trabajo en dejarlo todo medianamente funcional, por ello tuve que descartar esta opción por no ser viable en tiempo. Desafortunadamente, Unity3D mejoró dichas opciones para introducir CI, pero al estar con el proyecto tan avanzado y ante la complicación de realizar test automatizados, era totalmente inviable introducirlo a esas alturas.

Control de calidad del código

Desde el tribunal de calificación se recomienda utilizar una herramienta para analizar la calidad de código en los TFG y en este proyecto se planteó utilizar una de las herramientas recomendadas: SonarQube [43]. Por desgracia en Unity existe el mismo problema que con los test automáticos, y es que SonarQube no reconoce las clases propias de Unity, haciendo inservible la herramienta por los numerosos falsos positivos lanzados. No obstante, Visual Studio cuenta con un analizador de código estático que genera unas métricas del código del proyecto, permitiendo ver aspectos como la mantenibilidad, la complejidad ciclomática, la profundidad de herencia o las líneas del código [31]. No es un análisis tan extenso y detallado como el de SonarQube, de hecho no se puede ver nada más allá de los resultados, pero al menos son un mínimo de datos con los que mirar la salud del código realizado[4].

4.8. Plugins de Unity

Los plugins de Unity3D, o assets externos, son herramientas de terceros normalmente que se pueden incluir en el proyecto para potenciar alguna funcionalidad o añadir alguna herramienta extra al kit proporcionado por Unity, así como modelos, texturas, librerías de clases, etc. Estos plugins pueden ser gratuitos o de pago, y se adquieren a través de la Asset Store [51], aunque dependiendo de la licencia que tengan algunos plugins se pueden distribuir por otros medios externos.

Sqliter

Este proyecto requería de una base de datos, no muy extensa ni compleja, cuyo acceso fuese de rápido acceso en un dispositivo móvil y que no complicase el desarrollo. Por ello se decidió usar Sqlite [46], ya que es una base de datos que cumple con todo lo anterior. El problema es que Unity en si mismo no tiene ninguna opción o clase de acceso a una base de datos, y hacer una

[4]En los materiales adjuntos se podrá encontrar una hoja de cálculo con los datos generados por este análisis para el código propio del proyecto.

implementación desde cero iba a ser muy costosa en tiempo, por lo que se recurrió a un plugin: Sqliter [29]. Este plugin añade las clases y métodos básicos para utilizar una base de datos de Sqlite en Unity, y tras modificarlo para adaptarlo a este proyecto, se han ampliado algunas de sus funciones 5.4.

Textmesh Pro

Textmesh Pro [52] es un plugin que amplia las escasas opciones de personalización de las interfaces de usuario que proporciona Unity, permitiendo así mejorar la estética de las interfaces del proyecto.

PolyWorld: Woodland Low Poly Toolkit

PolyWorld [62] es un plugin que ayuda enormemente a la creación de entornos y elementos *low poly* en Unity. Permite generar mallas de pocos polígonos de terrenos generados por el propio Unity con bastantes opciones de configuración para ello.

Extended colliders 3D

Extended colliders 3D [42] es un plugin que amplia las formas básicas de las mallas de colisiones que proporciona Unity. No solo permite definir más formas, sino también editar algunas de sus características para dar más flexibilidad a la hora de tener mallas de colisiones con un número pequeño de polígonos.

Aspectos relevantes del desarrollo del proyecto

5.1. Gitflow y Unity

Se inició el proyecto con la idea de utilizar la metodología Gitflow[5] desde el principio con idea de tener bien estructurada cada funcionalidad que se fuese a añadir para permitir cambios y correcciones rápidas en dichas funcionalidades, así como permitir un flujo constante de versiones según avanzase el proyecto. No hubo mucho problema al principio cuando la mayoría de los elementos que se editaban eran los ficheros de código y estos no daban ningún problema al aplicar esta metodología. El problema empezó al iniciar los cambios en las escenas[6] y en los objetos[7] pertenecientes a estas, ya que el formato en el que se guardan las escenas es en un fichero *yml* en el que se guardan todas las propiedades y contenidos de las mismas. Al realizar cambios en una escena, tanto de sus propiedades como de los objetos de esta, cambia el fichero *yml*, generando cambios que se tienen que llevar al control de versiones. Si en el control de versiones tenemos dos ramas que trabajan sobre la misma escena, algo que suele ocurrir frecuentemente al incorporar nuevas funcionalidades, se producirán problemas al fusionar dichas ramas, ya que habrá dos versiones del mismo archivo con cambios muy dispares y, lamentablemente, la herramienta propia del control de versiones no puede fusionar los cambios de forma automática, y los cambios son poco o nada claros en algunos casos como para ser leídos, entendidos y fusionados a mano por parte de una persona. Por este motivo, y a falta de herramientas adecuadas para poder realizar este tipo de fusiones constantemente sin que fuesen bastante tediosas de realizar, se optó por excluir Gitflow del proyecto, utilizando desde ese momento una única rama de desarrollo y otra rama para versiones estables.

[5] Ver sección "Gitflow" del apartado 4.1.
[6] Ver sección "Escenas" del apartado 3.3.
[7] Ver sección "GameObjects" del apartado 3.3.

5.2. Fragmentación en los dispositivos Android

La aplicación creada está desarrollada para dispositivos que usen Android. Esto puede ser un problema a la hora de distribuir un juego y es algo que se suele tener en cuenta en este tipo de desarrollos. En este caso Daydream[8] solo funciona en versiones iguales o superiores a Nougat (7.0), versión que, tal y como se puede comprobar en la tabla 5.1 o en el gráfico de versiones 5.11, actualmente solo está en un 23 % de los dispositivos que usan Android. A esto, hay que sumarle la limitación de hardware que también impone Google sobre los dispositivos que pueden usar Daydream [20], siendo todos estos dispositivos de alta gama, y que hace falta el sistema de Daydream View[9] para la aplicación. Todo esto en conjunto hace que al desarrollar una aplicación de estas características se tenga en cuenta el mercado y la disponibilidad de usuarios finales que podrán usar el producto, ya que como se puede comprobar hay una serie de requisitos muy concretos y estrictos a cumplir.

Versión	Nombre	API	Distribución
2.3.3 - 2.3.7	Gingerbread	10	0.3 %
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4 %
4.1.x	Jelly Bean	16	1.7 %
4.2.x		17	2.2 %
4.3		18	0.6 %
4.4	KitKat	19	10.5 %
5.0	Lollipop	21	4.9 %
5.1		22	18 %
6.0	Marshmallow	23	26.0 %
7.0	Nougat	24	23.0 %
7.1		25	7.8 %
8.0	Oreo	26	4.1 %
8.1		27	0.5 %

Tabla 5.1: Distribución de las versiones de Android en el mercado a fecha 16/04/2018. Fuente: Android Developers Dashboard [10].

5.3. Rendimiento y pruebas en dispositivos móviles

Durante el desarrollo del proyecto se ha tenido especial cuidado con tener un buen rendimiento general de la aplicación en dispositivos móviles para tener

[8] Ver sección 3.5.
[9] Ver sección 3.5

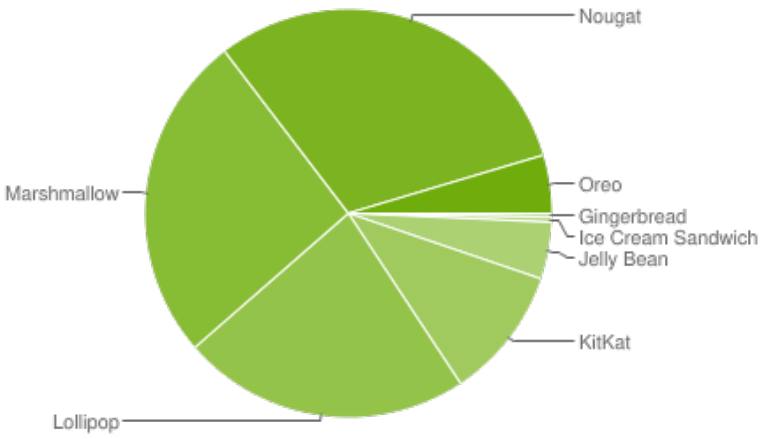

Figura 5.11: Gráfico de la fragmentación de las versiones de Android en el mercado a fecha 16/04/2018 [10].

una buena experiencia de usuario y evitar problemas como mareos, vómitos y malestar general en el usuario. Una de las particularidades de este aspecto es que todas las optimizaciones realizadas han sido teóricas, ya que no se ha contado con el dispositivo de pruebas hasta casi el final del proyecto. Esto ha hecho muy difícil calcular el impacto real de los cambios en el rendimiento. Un ejemplo de mejora teórica de rendimiento puede ser la eliminación del pintado en cámara de forma manual de los objetos que están en el angular de la cámara (ojos del jugador) pero que se encuentran a una distancia considerable como para que sean útiles al jugador (ver figura 5.12). Los objetos siguen estando ahí, pero la cámara no los pinta en la pantalla y el jugador no los ve, por lo que se gana rendimiento teórico gráficamente, aunque se perderá rendimiento por otro lado al tener almacenados en memoria más objetos de los estrictamente necesarios en ese momento.

Figura 5.12: Ejemplo de eliminación del pintado en cámara por distancia [37].

5.4. Modelos 3D y base de datos

A lo largo del proyecto se han ido incorporando nuevos modelos 3D de las setas y se han ido actualizando otros. Con cada cambio en los modelos, era obligatorio actualizar los elementos que hacían referencia a los modelos viejos, haciendo de esto una tarea un poco pesada y compleja al tener que hacerlo inicialmente de forma manual en muchos elementos. Más adelante se descubrió la forma de hacer que estos cambios tuviesen menos carga manual, aunque no ha sido posible dejarlo todo automatizado. Por ejemplo la importación de modelos en Unity hace referencia a nombres internos del modelo 3D, solo actualizables desde las aplicaciones de diseño, y varios componentes de las setas necesitan referencias a estos nombres, haciendo obligatorio que se sepa de antemano estos nombres o en su defecto la nomenclatura que se ha seguido con ellos. Lamentablemente, en los modelos de arte no se ha seguido una nomenclatura concreta existiendo numerosos elementos idénticos, haciendo imposible realizar esta automatización. A lo largo del proyecto también se han introducido nuevas especies de setas en la base de datos inicial, obligando a actualizar las referencias a los datos y teniendo que tener en cuenta posibles cambios de formato o de extensión en todos los elementos de presentación de los datos. Además, se ha dejado reflejada en la base de datos una estructura suficiente para poder ampliar la localización de la aplicación a varios idiomas en alguna iteración futura de la aplicación.

5.5. Acceso a los datos, contenedores de datos y automatización

Uno de los elementos clave en este proyecto es el uso de los datos que vienen dados por la base de datos: el jugador necesita poder ver los datos de la seta que acaba de coger. Este requisito hizo que al principio del proyecto hubiese que decidir cómo leer esos datos, bien realizando múltiples llamadas a la base de datos cada vez que fuese necesario algún dato, o bien haciendo una única lectura al principio y almacenando todos los datos necesarios en memoria. Ambas implementaciones tenían sus ventajas e inconvenientes. El realizar accesos a la base de datos cuando se necesitan datos sería la forma ideal para una lectura rápida y actualizada de datos, ya que si estos han cambiado desde la ejecución inicial los cambios se van a ver reflejados en algún momento si se vuelve a leer el dato cambiado. Por otro lado, esta implementación supone mantener abierta una conexión a la base de datos constantemente y perder tiempo, por poco que sea, en cada acceso. La otra forma, leer una única vez y guardarlo todo en memoria, permite tener a mano todos los datos que sean necesarios y elimina la necesidad de mantener una conexión abierta constante. Por otro lado, esto tiene el inconveniente de que se pierde mucha memoria si la base de datos es muy grande y de que si los datos se actualizan en mitad

de la ejecución, no hay posibilidad de acceder a los nuevos datos.

Valorando estos aspectos, se decidió optar por la segunda opción por el temor a que mantener una conexión abierta tuviese un impacto muy negativo en el rendimiento o que al acceder a los datos se produjesen molestos picos en el rendimiento, algo bastante molesto en Realidad Virtual[10]. Para no tener graves problemas de memoria[11], se recurrió a los *ScriptableObjects*[12] por su utilidad a la hora de cuidar el uso de memoria, además de ser ideales en este caso: aunque tengamos cien setas de una especie a la vez en el nivel, la información de la especie va a ser la misma.

Otro beneficio de usar los *ScriptableObjects* es que se pueden modificar los datos que contengan en tiempo de ejecución sin que se produzcan errores, lo que permite que cualquier objeto de Unity pueda utilizar la información que contiene, aunque esté vacía, y modificarla cuando sea necesario sin ningún problema. Esto da flexibilidad en el desarrollo, pues se pueden dejar definidos en el código o en el editor todos los accesos a las propiedades de estos contenedores y modificarlos cuando sea conveniente, bien cuando se están desarrollando y creando los niveles por ejemplo, como cuando la aplicación se encuentra en ejecución. En este caso, para reducir el impacto de rendimiento, se optó por generar los contenedores previamente.

El último problema de generar los contenedores en tiempo de desarrollo, es cuáles generar o cuántos generar: todo lo que se genere tendrá su peso en la memoria. La base de datos usada contiene muchas especies de setas, muchas más de las que se ha podido integrar en el proyecto, por lo que es absurdo generar contenedores de algo que no se va a utilizar. Por ello se decidió generar solo los contenedores de las setas que realmente puedan ser usadas[13] y se intentó automatizar la creación total de dichos contenedores, siendo extensible a la creación de los componentes necesarios para poder dejar definida una plantilla para utilizar la seta en el proyecto. Hubo un éxito parcial, ya que se pueden crear contenedores de las setas individualmente de forma automática sabiendo el nombre de la seta, pero no se ha conseguido automatizar la creación de todas las setas que puedan llegar a usarse en el proyecto, así como tampoco se ha conseguido la automatización de la creación de componentes, como ya se ha explicado en el apartado anterior.

[10] Ver sección "Rendimiento" del apartado 3.4

[11] Destacar que el proyecto se ejecuta en un dispositivo móvil, donde la memoria está más limitada

[12] Se usará ScriptableObjects o contenedores de datos indistintamente, haciendo referencia al mismo concepto. Ver sección "ScriptableObjects" del apartado 3.3

[13] Para saber si una seta podía ser usada o no en el proyecto el elemento limitante es el modelo 3D de dicha seta: si existe su modelo en el proyecto, se puede llegar a usar.

5.6. Complejidad de las mallas de colisiones y su impacto en el rendimiento

En las últimas semanas del desarrollo, cuando se pudieron hacer pruebas ya con el dispositivo móvil, se descubrió que la aplicación, durante su ejecución, provocaba fallos internos en el teléfono que causaban inestabilidad en ciertos elementos de la aplicación, así como tiempos de carga excesivamente grandes. Estos fallos eran debidos a que las setas utilizan una malla de colisiones [65] demasiado compleja para detectar las interacciones físicas con el entorno y para que el usuario pueda interactuar con las setas en si. Resulta que Unity tiene un límite en el número de polígonos que puede tener una malla de colisiones para una aplicación Android, siendo 255 este límite, y se estaban utilizando mallas de miles de polígonos. Este fallo no era visible en las ejecuciones en el ordenador y no se descubrió hasta que se vieron fallos en la ejecución en el dispositivo y se activó el *logcat* del SDK de Android [19] para monitorizar en tiempo real el registro interno generado. Unity por defecto genera la malla más ajustada posible al modelo que se esté utilizando por lo que si el modelo tiene alto número de polígonos, como es el caso, la malla tendrá también un alto número de polígonos.

Conociendo la existencia de este límite y de cómo Unity genera estas mallas, la primera opción planteada para solucionar este problema era repetir los modelos bajando el número de polígonos y con ello el nivel de detalle de los modelos. Esta opción se descartó enseguida por la proximidad de la fecha límite para el desarrollo del proyecto, siendo además la opción más costosa en tiempo. La otra opción planteada era utilizar las mallas de formas simples que proporciona Unity, que aunque no tienen la misma fidelidad a la hora de la interacción con el entorno por tener formas simples, tienen tan poco coste computacional que se pueden colocar varias para intentar que la malla combinada se asemeje a la forma real del modelo. El problema de estas mallas simples es que no hay formas que por si solas se asemejen a la forma variada de las setas y sería necesario combinar demasiadas mallas simples para conseguir un efecto aceptable. Es por eso que finalmente se decidió consultar la *Asset Store* de Unity para encontrar el plugin *Extended Colliders 3D* [42]. Este plugin permite generar el mismo tipo de mallas con formas simples más variadas y personalizables que las propias que vienen por defecto en el editor.

Utilizando este plugin finalmente se bajan las mallas de colisiones de miles de polígonos a menos de 10 polígonos por malla con un acabado bastante aceptable en términos de fidelidad, lo que no solo elimina el error, sino que además mejora enormemente el rendimiento de la aplicación en las cargas, bajando el tiempo de carga de los niveles de varios minutos a solo unos pocos segundos con cientos de setas en el nivel. En la figura 5.13 se puede apreciar la comparación entre la malla de colisiones generada por defecto por Unity y la nueva malla de colisiones utilizada.

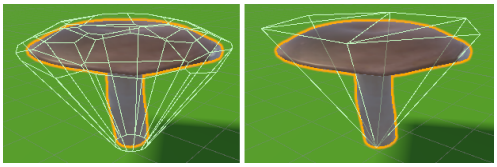

Figura 5.13: Comparación entre la malla de colisiones por defecto y la malla de colisiones generada por el plugin Extended Colliders 3D

5.7. Publicación de la aplicación en Google Play

Aunque inicialmente no hay idea de comercializar la aplicación resultante, se ha publicado una versión alfa de la aplicación en la tienda digital de aplicaciones *Google Play* [17] para facilitar su distribución. Esta versión es privada y por tanto no está accesible públicamente salvo por invitación del propietario, pero por todo lo demás es exactamente igual a cualquier aplicación en producción.

El proceso para publicar la aplicación ha requerido de varios pasos, siendo el inicial tener un fichero apk de una versión estable con la depuración desactivada, ya que por motivos de seguridad Google no permite aplicaciones en su tienda que tengan habilitada esta opción. Después se ha tenido que realizar una ficha de la aplicación, siendo necesarias capturas de la misma en funcionamiento y varias imágenes publicitarias, así como una imagen estereoscópica o de 360º para que la aplicación pueda ser catalogada como aplicación de Daydream. En la figura 5.14 se puede ver el resultado de la ficha de la aplicación en la tienda. Una curiosidad sobre la publicación es que hay que marcar la aplicación como candidata a *Daydream app*, el equipo de Google revisa que cumpla con los requisitos de calidad [14] y pasado un tiempo aprueban o rechazan la aplicación para que aparezca como compatible con Dayream en la tienda. En este caso la aplicación ha pasado los requisitos de calidad, como se puede ver en la figura 5.15, y si la versión en alfa se pasase a producción aparecería listada en la tienda como otra aplicación más.

Otro detalle curioso es que al subir una versión a *Google Play* desde la plataforma se pueden realizar pruebas en dispositivos reales compatibles con la aplicación antes del lanzamiento de la misma, con el fin de localizar bloqueos, problemas de pantalla o vulnerabilidades de seguridad. En este caso, al subir la versión alfa se decidió ejecutar las pruebas en dos móviles Google Pixel, uno con Android Nougat (7.1) y el otro con Android Oreo (8.0). Las pruebas realizadas eran pruebas Robo [21], aunque estas pruebas en las aplicaciones de Unity tienen la limitación de que no prueban más allá de la primera pantalla, pero parecía una forma interesante de probar el rendimiento de inicio de la aplicación en otros dispositivos. Los resultados se pueden ver en la figura 5.16 y es interesante comprobar como el tiempo de inicio ha sido mejor en el *Pixel*

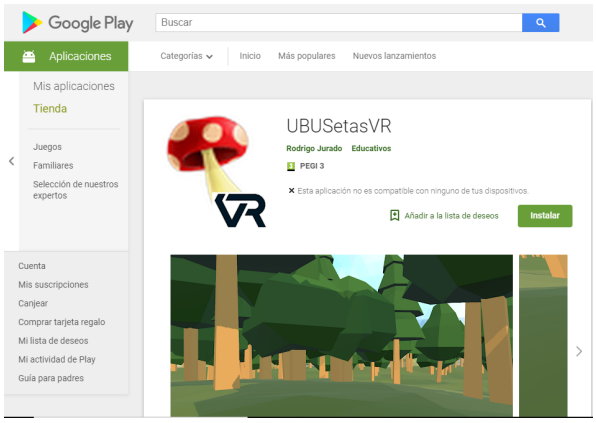

Figura 5.14: Ficha de la aplicación en Google Play.

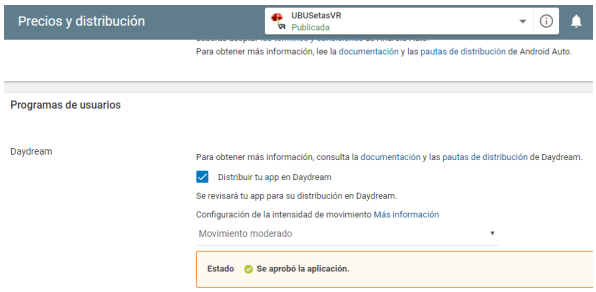

Figura 5.15: Aprobación de la aplicación como *Daydream app*.

con Nougat, aunque haya consumido más recursos que el de Oreo.

Figura 5.16: Resultados de la ejecución de pruebas Robo en Google Play.

Trabajos relacionados

6.1. Prototipo de simulador de carretilla elevadora

Me parece adecuado como primera referencia este prototipo de carretilla elevadora [33] por su gran similitud con este proyecto, así como la cercanía en tiempo de su presentación, que fue a principio de 2018. En el proyecto se puede ver el uso de Unity para desarrollar un prototipo de entrenamiento que usa Realidad Virtual y que hace hincapié en el realismo del sistema de físicas utilizadas. Destacar que dicho proyecto tiene como objetivo dispositivos fijos de altas prestaciones y coste, en comparación con este proyecto, el cual se centra en dispositivos móviles de prestaciones mucho más bajas y un coste significativamente menor, siendo claros ejemplos de las dos caras de una misma moneda, o de como aplicar una Realidad Virtual similar en dispositivos totalmente opuestos.

6.2. UBUSetas

En este otro proyecto [8], también presentado en 2018, se ha realizado una aplicación para dispositivos móviles que se centra en aplicar *Machine Learning* para reconocer y clasificar diferentes especies de setas usando fotografías suministradas por el usuario. El proyecto usa una base de datos Sqlite con los datos de las especies, la cual ha sido reutilizada para este proyecto por el gran número de entradas disponibles.

6.3. Creación de un videojuego y un bot inteligente para el mismo

En este otro proyecto [40], presentado en 2017, se utiliza también Unity como herramienta para crear un videojuego que permita a un algoritmo aprender

a jugar al mismo a partir de las partidas de otros jugadores humanos, utilizando *Machine Learning* en el proceso.

6.4. Have Fung

En 2015 EON Reality realizó una aplicación de Realidad Aumentada para dispositivos móviles llamada "Have Fung" [39] que tenía finalidad educacional y permitía conocer mejor la comestibilidad de algunas especies concretas de setas. Es un proyecto visualmente muy similar aunque no hace uso de la Realidad Virtual y aparentemente lo único que queda de referencia a el es una nota de prensa y un vídeo de ejemplo de cómo funcionaba.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones del proyecto

El proyecto se inició con la intención de desarrollar un videojuego completo de Realidad Virtual desde cero, con buen rendimiento en los dispositivos móviles y muy modular, con idea de que si en un futuro se quiere ampliar, por ejemplo con más niveles, se contase con las herramientas necesarias para ello.

Echando la vista atrás, se puede considerar que el proyecto resultante ha sido bastante diferente a como lo había planeado inicialmente y he cambiado bastantes elementos y planteamientos a lo largo de todo el desarrollo, bien por decisiones propias, bien por inconvenientes o problemas surgidos, siendo uno de los más determinantes la recepción tardía del hardware necesario para las pruebas.

Durante el desarrollo no me he conformado con aplicar solo lo aprendido en la carrera, sino que he intentado aplicar técnicas nuevas y he probado a experimentar, tanto en el código como en el diseño en el propio Unity, algo a lo que no estoy muy acostumbrado. Todo el trabajo de documentación realizado durante este proyecto ha sido extenso, del cual he aprendido muchos conceptos y he podido indagar más en el estado del arte de ciertos elementos del mundo móvil, que junto con el proceso de desarrollo, me será de mucha utilidad en un futuro puesto que tengo intención de enfocar mi experiencia profesional en este campo. También ha sido un proyecto donde he podido aplicar mucha autocrítica, pues he podido apreciar carencias en ámbitos que antes de empezar hubiese asegurado que manejaba mejor, al igual que he tenido varios momentos de disconformidad con el propio trabajo que han desembocado en exigirme más con mejores resultados.

No obstante y pese a todo, estoy bastante contento con el resultado obte-

nido, ya que creo que se ha conseguido cumplir con gran parte de los objetivos planeados para este proyecto, sin olvidar que hay bastante margen de mejora del producto final. Destacar finalmente que, pese a tener conocimientos previos sobre Realidad Virtual y haber tenido la posibilidad de probar potentes equipos destinados a la misma, me ha sorprendido gratamente la experiencia de probar la Realidad Virtual en un dispositivo móvil ya que no me imaginaba que se pudiese encontrar en un estado tan bueno actualmente y creo que habrá muchas más opciones de proyectos futuros siguiendo esta línea de trabajo.

7.2. Líneas de trabajo futuras

- **Diseño de niveles.** El estilo de los niveles se puede mejorar mucho y se podría ampliar el nivel de detalle de los mismos para que no estén tan vacíos. Además, se podría cambiar también el aspecto estético de los niveles y darle otro acabado.
- **Diseño de interfaz.** El estilo de los paneles de información de las setas y el panel de fin de juego son muy mejorables en cuanto a su estilo.
- **Experiencia de usuario.** No hay casi elementos que orienten al usuario sobre si lo que está haciendo está bien o está mal o que le den pistas visuales o auditivas de lo que está ocurriendo en el juego. Se podría mejorar bastante en este aspecto.
- **Sonidos.** Si bien es cierto que el juego cuenta con algún sonido ambiental, no hay apenas sonidos o música, y los sonidos en Realidad Virtual son muy importantes para la inmersión total, aspecto que se podría (y debería) mejorar en un futuro.
- **Controles.** El mando de Daydream es bastante escaso en posibilidades de control, pero se podría dar un revisado a los controles implementados en este proyecto, ya que puede que sean algo complicados para un recién llegado a la Realidad Virtual.
- **Tutorial.** Siguiendo con el apartado anterior, el que los controles sean complicados es también un efecto de que no haya un nivel de tutorial que te explique bien dichos controles.

Bibliografía

[1] James R. Trott Alan Shalloway. Design patterns explained: A new perspective on object-oriented design, 2004.

[2] Amazon. Lumberyard. https://aws.amazon.com/lumberyard/, 2017.

[3] Software Freedom Conservancy. Git. https://git-scm.com/, 2017.

[4] Vincent Driessen. A successful git branching model. http://nvie.com/posts/a-successful-git-branching-model/, 2010.

[5] Asociación española de empresas productoras y desarrolladoras de videojuegos y software de entretenimiento. *Libro Blanco del Desarrollo Español de Videojuegos*. DEV, 2016.

[6] P. Fuchs. *Virtual Reality Headsets - A Theoretical and Pragmatic Approach*. CRC Press, 2017.

[7] Epic Games. Unreal engine. https://www.unrealengine.com/en-US/what-is-unreal-engine-4, 2017.

[8] Adrián Antón García. *UBUSetas 1.0*. PhD thesis, Grado Ingeniería Informática. Universidad de Burgos, Feb 2018.

[9] GitHub. Github - about site. https://github.com/about, 2018.

[10] Google. Android distribution dashboard. https://developer.android.com/about/dashboards/, 2018.

[11] Google. Cardboard. https://vr.google.com/cardboard/, 2018.

[12] Google. Daydream. https://vr.google.com/daydream/, 2018.

[13] Google. Daydream - google developers site. https://developers.google.com/vr/discover/daydream, 2018.

[14] Google. Daydream-app quality requirements. https://developers.google.com/vr/distribute/daydream/app-quality, 2018.

[15] Google. Daydream degrees of freedom. https://developers.google.com/vr/discover/degrees-of-freedom, 2018.

[16] Google. Daydream view. https://vr.google.com/intl/es_es/daydream/smartphonevr/, 2018.

[17] Google. Google play. https://play.google.com/store, 2018.

[18] Google. Google vr - google developers site. https://developers.google.com/vr/, 2018.

[19] Google. Logcat command-line tool. https://developer.android.com/studio/command-line/logcat, 2018.

[20] Google. Phones compatible with daydream. https://vr.google.com/daydream/smartphonevr/phones/, 2018.

[21] Google. Robo test - firebase test lab. https://firebase.google.com/docs/test-lab/android/robo-ux-test, 2018.

[22] Indie Film Hustle. Frame rates: Do not shoot 24fps. https://indiefilmhustle.com/frame-rates-24fps/, 2017.

[23] VR Lens Lab Jay. Field of view for virtual reality headsets explained. https://vr-lens-lab.com/field-of-view-for-virtual-reality-headsets/, 2016.

[24] J. Jerald. *The VR Book: Human-Centered Design for Virtual Reality*. ACM Books. Association for Computing Machinery and Morgan & Claypool Publishers, 2015.

[25] J. Jerald. *The VR Book: Human-Centered Design for Virtual Reality*, chapter 25.2.3. ACM Books. Association for Computing Machinery and Morgan & Claypool Publishers, 2015.

[26] Jetbrains. Rider. https://www.jetbrains.com/rider/, 2017.

[27] Filipp Keks. Why game developers are afraid of test automation? http://blog.filippkeks.com/2016/11/21/why-game-developers-are-afraid-of-test-automation.html, 2016.

[28] Michael Klappenbach. Understanding and optimizing video game frame rates. https://www.lifewire.com/optimizing-video-game-frame-rates-811784#mntl-sc-block_1-0-19, 2018.

[29] OuijaPaw Games LLC. Sqliter. https://www.assetstore.unity3d.com/en/#!/content/20660, 2014.

[30] Microsoft. Visual studio. https://www.visualstudio.com/, 2017.

[31] Microsoft. Code metrics values. https://msdn.microsoft.com/en-us/library/bb385914.aspx, 2018.

[32] MonoDevelop. Monodevelop - main site. https://www.monodevelop.com/, 2018.

[33] Rodrigo Varga Moreno. *Prototipo de simulador de carretilla elevadora*. PhD thesis, Grado Ingeniería Informática. Universidad de Burgos, Feb 2018.

[34] Lukasz Paczkowski. Replacing monodevelop-unity with visual studio community starting in unity 2018.1. https://blogs.unity3d.com/es/2018/01/05/discontinuing-support-for-monodevelop-unity-starting-in-unity-2018-1/, 2018.

[35] Rodrigo Jurado Pajares. Github - ubusetasvr. https://github.com/Kiszaner/2017_SetasVR_TFG, 2017.

[36] Rodrigo Jurado Pajares. Ubusetasvr - closed alpha. https://play.google.com/apps/testing/es.ubu.eps.admirable.ubusetasvr, 2018.

[37] Unity Performance. Unity culling techniques. http://unityperformance.com/guide/culling, 2016.

[38] E. Preisz. *Video Game Optimization*, chapter 2, pages 32–35. Books 24x7 IT PRO. Course Technology, 2010.

[39] EON Reality. Eon reality releases "have fung", an educational augmented reality mushroom app. https://www.eonreality.com/press-releases/eon-reality-releases-have-fung-an-educational-augmented-reality-mushroom-app/, 2015.

[40] Pablo Alejos Salamanca. *Creación de un videojuego y un bot inteligente para el mismo*. PhD thesis, Grado Ingeniería Informática. Universidad de Burgos, Jul 2017.

[41] Scrum.org. What is scrum? https://www.scrum.org/resources/what-is-scrum, 2017.

[42] BATTENBERG SOFTWARE. Extended colliders 3d. https://assetstore.unity.com/packages/tools/physics/extended-colliders-3d-46035, 2018.

[43] Sonar Source. Sonarqube. https://www.sonarqube.org/, 2017.

[44] VR Source. How does virtual reality work? https://vrsource.com/virtual-reality-work-3788/, 2016.

[45] Sourcetree. Sourcetree - main site. https://www.sourcetreeapp.com/, 2018.

[46] Sqlite. Sqlite. https://www.sqlite.org/index.html, 2000.

[47] Andrew Lucas Studios. A quick guide to degrees of freedom in virtual reality. https://lucas-studios.com/quick-guide-degrees-of-freedom-virtual-reality-vr/, 2018.

[48] SuperData. Infographic: Virtual reality on the rise. https://www.superdataresearch.com/infographic-virtual-reality-on-the-rise/, 2017.

[49] NeuroDigital Technologies. Gloveone kickstarter funding campaign. https://www.kickstarter.com/projects/gloveone/gloveone-feel-virtual-reality, 2015.

[50] Unity Technologies. Unity joins the .net foundation. https://blogs.unity3d.com/es/2016/04/01/unity-joins-the-net-foundation/, 2016.

[51] Unity Technologies. Asset store. https://assetstore.unity.com/, 2017.

[52] Unity Technologies. Textmesh pro. https://www.assetstore.unity3d.com/en/#!/content/84126, 2017.

[53] Unity Technologies. Unity3d. https://unity3d.com/, 2017.

[54] Unity Technologies. Creating and using scripts. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html, 2018.

[55] Unity Technologies. .net profile support. https://docs.unity3d.com/Manual/dotnetProfileSupport.html, 2018.

[56] Unity Technologies. Unity compatible platforms. https://unity3d.com/es/unity/features/multiplatform, 2018.

[57] Unity Technologies. Unity components. https://docs.unity3d.com/Manual/UsingComponents.html, 2018.

[58] Unity Technologies. Unity game objects. https://docs.unity3d.com/Manual/GameObjects.html, 2018.

[59] Unity Technologies. Unity scenes. https://docs.unity3d.com/Manual/CreatingScenes.html, 2018.

[60] Unity Technologies. Unity scriptableobject fundamentals. https://docs.unity3d.com/Manual/class-ScriptableObject.html, 2018.

[61] Unity Technologies. Unity subscriptions. https://store.unity.com/, 2018.

[62] Quantum Theory. Polyworld: Woodland low poly toolkit. https://assetstore.unity.com/packages/3d/environments/landscapes/polyworld-woodland-low-poly-toolkit-18572, 2018.

[63] THINHHB. Using scriptableobject to store game configurations. https://thinhhb.wordpress.com/2016/05/01/unity3d-using-scriptableobject-to-store-game-configurations/, 2016.

[64] Trello. Trello. https://trello.com/home, 2017.

[65] UnityTips. How to fix unity convex colliders. https://www.unity3dtips.com/convex-hull-has-more-than-255-polygons/, 2018.

[66] Wikipedia. Frame rate. https://en.wikipedia.org/w/index.php?title=Frame_rate&oldid=837343783, 2018.

[67] Wikipedia. Game engine. https://en.wikipedia.org/w/index.php?title=Game_engine&oldid=840124390, 2018.

[68] Wikipedia. Htc vive. https://en.wikipedia.org/w/index.php?title=HTC_Vive&oldid=840610031#Hardware_and_accessories, 2018.

[69] Wikipedia. Oculus rift headset. https://en.wikipedia.org/w/index.php?title=Oculus_Rift&oldid=838384295#Controllers, 2018.

[70] Wikipedia. Six degrees of freedom. https://en.wikipedia.org/w/index.php?title=Six_degrees_of_freedom&oldid=837291423, 2018.

[71] Wikipedia. Three degrees of freedom. https://en.wikipedia.org/w/index.php?title=Degrees_of_freedom_(mechanics)&oldid=836184728, 2018.

[72] Wikipedia. Virtuix omni treadmill. https://en.wikipedia.org/w/index.php?title=Virtuix_Omni&oldid=842260648, 2018.

[73] Xinreality. Degrees of freedom. https://xinreality.com/mediawiki/index.php?title=Degrees_of_freedom&oldid=9792, 2016.

[74] ZenHub. Zenhub. https://www.zenhub.com/, 2017.