TELECOMUNICACIÓN

Campus Sur

POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Mejora de la experiencia y del contenido del videojuego terapéutico “Phiby’s Adventures 3D”

AUTOR: Laura Álvaro Gil

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

TUTOR: Martina Eckert

DEPARTAMENTO: Ingeniería Audiovisual y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: José Antonio Sánchez Fernández

TUTOR: Martina Eckert

SECRETARIO: Enrique Rendón Angulo

Fecha de lectura:

Calificación:

El Secretario,

Agradecimientos

A todas esas personas que me han hecho crecer durante estos años de carrera.

A mi familia, por creer incondicionalmente en mí y en mi éxito en cada aventura que emprendo por muy dura que pueda parecer o los baches que haya.

En especial a mi madre, por ser mi pilar inquebrantable en esos momentos en los que parece que todo se derrumba, por aportarme calma y serenidad cuando más lo necesito.

Resumen

Este proyecto de fin de grado consiste en la revisión y mejora del juego terapéutico en desarrollo “Phiby’s Adventures 3D” para personas con movilidad reducida o en rehabilitación, buscando una mejora de la experiencia del jugador a través de la introducción de nuevos contenidos que motiven al jugador a continuar con la historia y hacer que su sesión de juego resulte amena y divertida a la vez que se cumplen los objetivos de rehabilitación o movilidad designados.

Para el desarrollo de este proyecto se hace uso de una cámara de detección de captación 3D y una plataforma web usada por los terapeutas para controlar la dificultad de los ejercicios del juego. El proyecto se desarrolla en Unity y en lenguaje de programación C#.

Este trabajo de fin de grado parte de la última versión existente a la fecha de comienzo de este proyecto. Según el análisis de esa versión se definen unos objetivos a desarrollar durante la duración del proyecto de fin de grado, los cuales se detallan en este documento por si en el futuro fuera de ayuda conocer su desarrollo más a fondo.

Para dotar de dinamismo al juego, se ha mejorado una parte de la historia la cual puede resultar tediosa para el jugador al no requerir de mucha interacción por su parte. A partir de esta mejora de la historia se ha creado un nuevo ejercicio de movilidad para incluir más dinamismo en la presentación del contenido del juego. Se han introducido algunas cinemáticas en varios puntos del desarrollo de la historia para dar un descanso al jugador y mejorar el hilo narrativo; además, se ha creado un sistema de reproducción de cinemáticas totalmente funcional por si en un futuro se deciden incluir más cinemáticas y se ha documentado el proceso de grabación de cinemáticas usando sólo herramientas de Unity. Por último, se ha revisado parte de la lógica ya implementada en la última versión existente para mejorar el rendimiento del juego.

Abstract

This final degree project consists in the revision and improvement of the therapeutic game “Phiby’s Adventures 3D” for people with reduced mobility or in rehabilitation, seeking to enhance player’s experience through the addition of new content that will boost player’s motivation to continue the game’s story and make their game session more enjoyable and fun and at the same time they can complete their rehabilitation or movement objectives.

A 3D movement detection camera is used for the development of this project, as well as a web platform for the therapists to control the difficulty of the game’s exercises. The project is developed in Unity and the programming language C#.

The development starts from the latest version of the game existing at the start date of this project. That version is analysed, and a set of objectives are set to develop during the duration of the final degree project, which are detailed in this document for possible more in-deep future references.

To provide the game with more dynamism a part of the story has been improved, because it resulted boring due to few interaction possibilities for the player. Taking advantage of this modifications of the story, a new mobility exercise has been created in order to present the content of the game more dynamically. In addition, various cinematics have been introduced in some points of the game’s story development to give the player a break and aid the narrative thread; moreover, a completely functional cinematic player has been implemented in the project, for if in the future it would be necessary to add more cinematics. The process of creation and recording of cinematics using only Unity tools has been properly documented. Finally, some parts of the logic already implemented in previous versions has been revised and improved in order to boost the game’s efficiency.

Acrónimos

GAMMA:	Grupo de Aplicaciones Multimedia y Acústica
CITSEM:	Centro de Investigación en Tecnologías del Software y Sistemas Multimedia para la Sostenibilidad.
SDK:	Software Development Kit
K2UM:	Kinect to Unity Middleware
NPC:	Non-Player Character
FMV:	Full Motion Video
HUD:	Heads-Up Display
FPS:	First Person Shooter

Índice de contenido

Índice de figuras

Índice de tablas

1. Introducción

Durante los últimos años el desarrollo de juegos serios con fines terapéuticos está creciendo constantemente. Además, la reciente mejora en el desarrollo de tecnologías de detección y captación de movimiento resulta en multitud de grupos de investigación como GAMMA (Grupo de Aplicaciones Multimedia y Acústica) (perteneciente al CITSEM (Centro de Investigación en Tecnologías del Software y Sistemas Multimedia para la Sostenibilidad)) que trabajan en el desarrollo de proyectos como el entorno Blexer (Blexer Exergames). Son muchos los alumnos que han contribuido a este proyecto participando tanto con prácticas como con el desarrollo de su proyecto de fin de grado, como es el caso.

Este proyecto de fin de grado es la continuación del juego terapéutico “Phiby’s Adventures 3D”, un proyecto que entra dentro del marco de desarrollo de Blexer y que empezó su desarrollo en el año 2018. Este juego, desarrollado en Unity, está orientado a pacientes jóvenes con movilidad reducida, y pretende amenizar las sesiones de terapia o rehabilitación haciendo que los movimientos sean capturados en el juego y se usen como una mecánica más. Esto se consigue al usar una cámara de captación 3D, concretamente la Kinect v2 de Microsoft. Al formar parte del proyecto Blexer, los ejercicios de “Phiby’s Adventures 3D” disponen de parámetros configurables por los terapeutas gracias a la existencia de comunicación con la página web Blexer-med, donde los terapeutas también pueden analizar los resultados obtenidos por sus pacientes durante las sesiones de juego.

Este proyecto de fin de grado persigue la revisión y mejora de “Phiby’s Adventures 3D”, buscando mejorar la experiencia del jugador introduciendo dinamismo y variación del contenido ya existente en el juego para obtener una experiencia amena, divertida y enriquecedora. Se busca que el jugador tenga incentivos para seguir jugando y no se aburra durante el transcurso de las sesiones de juego. Estas mejoras se han llevado a cabo junto con Juan Fuentes-Pila [1], alumno de la escuela que también ha llevado a cabo su proyecto de fin de grado desarrollando este videojuego.

2. Antecedentes y entorno del proyecto

La idea de que los videojuegos son meramente recreativos es, desde hace ya bastante tiempo, una idea un tanto obsoleta: pueden servir también como un medio para conseguir otros muchos objetivos; éstos son los llamados juegos serios. Uno de esos objetivos consiste en el uso de videojuegos y sus propiedades para personas con una discapacidad física. La tecnología no para de realizar avances en este campo y la rehabilitación y trabajo de la movilidad física se benefician de ello.

Los proyectos desarrollados con este fin son denominados videojuegos de simulación terapéuticos, pues a través de las características positivas de los videojuegos tradicionales (diversión, entretenimiento, motivación, …) favorecen el aprendizaje y la práctica de estrategias mientras el jugador realiza movimientos de rehabilitación o de ejercicio orientados a una movilidad reducida.

Una de las principales vías de investigación del grupo GAMMA (grupo reconocido por la Universidad Politécnica de Madrid perteneciente al CITSEM) trata la gamificación y el desarrollo de interfaces naturales e inteligentes para personas con discapacidad física. Trabajando con terapeutas diseñan y desarrollan videojuegos para la rehabilitación física, ejercicios de mantenimiento, etc., con la ayuda de las nuevas tecnologías como por ejemplo cámaras con captación de movimientos. Existen otros proyectos similares en el mercado como pude ser VirtualRehab 3.0 de la compañía Evolve [2] o la plataforma de software MIRA [3].

2.1. Entorno Blexer

Dentro de la investigación para la gamificación y el desarrollo de interfaces naturales para fines terapéuticos [4] que lleva a cabo el grupo GAMMA se encuentra el entorno de Blexer. Este entorno está comprendido por diferentes videojuegos enfocados, sobre todo, a niños y jóvenes con movilidad reducida de forma que puedan realizar el ejercicio físico que necesitan de una forma más amena y divertida. Estos videojuegos han sido desarrollados a lo largo del tiempo en colaboración con alumnos de la escuela.

El entorno cuenta con una plataforma de datos llamada Blexer-med [5]. Esta plataforma permite controlar y mantener un registro de los requisitos y resultados de los pacientes en cada uno de los videojuegos y fue desarrollada por Mónica Jiménez [6] y Alba Aguilar [7]. Estos datos son almacenados a través de Internet en un servidor web y en un servidor MYSQL.

De esta forma el terapeuta o supervisor puede adaptar los parámetros de cada videojuego para ajustar el esfuerzo que debe realizar el jugador. Además, se puede llevar un registro de los datos a lo largo del tiempo para poder ver la evolución del jugador: se registran tanto los fallos como los movimientos realizados correctamente, permitiendo ajustar de forma inmediata el ejercicio del jugador. Algunos parámetros podría ser el tiempo de realización del ejercicio o el número mínimo de movimientos, pero dependerán del juego los parámetros que se puedan ajustar. A la plataforma Blexer-med se puede acceder con tres perfiles: terapeuta (para controlar y ajustar los parámetros del jugador y ver su evolución), administrador de un centro y superadministrador (añadir y editar juegos y sus parámetros, crear nuevos administradores y terapeutas).

Como ya se ha mencionado anteriormente, estos juegos se basan en el uso de cámaras con captación de movimientos. En este caso se hace uso de la Kinect v2. de Microsoft [8]. Este dispositivo se basa en el reconocimiento de los movimientos del usuario y la identificación de su voz. De esta forma permite interactuar con aplicaciones usando movimientos corporales o comandos de voz, sin necesidad de otros periféricos como podrían ser un ratón o un teclado. Para poder usar este dispositivo es necesario disponer de un ordenador con una versión del sistema operativo Windows 8.1 o superior, y para poder desarrollar aplicaciones también es necesario instalar el paquete de desarrollo de software de Kinect para Windows (SDK (*Software Development Kit*) [9].

Para poder cargar y descargar datos del juego es necesario que el jugador disponga en el ordenador donde ejecuta el juego del middleware K2UM (*Kinect to Unity Middleware*) desarrollado por César Luaces [10]. A través de este software se realiza la comunicación entre el dispositivo de captación de movimiento y el juego, así como de la comunicación del juego (y sus parámetros) con la plataforma Blexer-med.

El funcionamiento del entorno Blexer puede verse esquematizado en la Figura 1, donde se representa la relación de todas las partes involucradas en dicho sistema. En la parte izquierda de la figura se encuentra el ordenador del paciente con su Kinect, su juego y su *middleware*. En la parte izquierda se encuentra por un lado los servidores usados para la comunicación y el guardado de datos, y por otra parte la plataforma Blexer-med utilizada por los terapeutas.

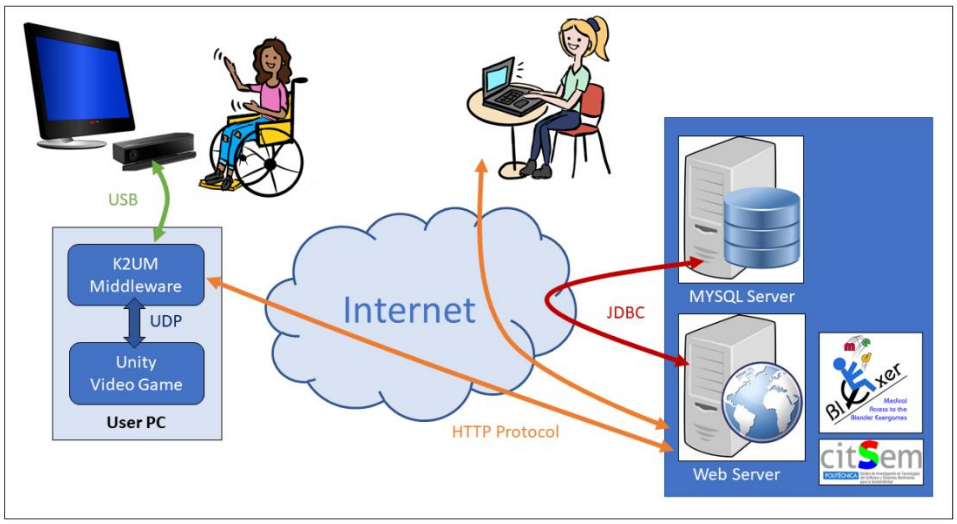

Figura 1. Esquema del sistema Blexer. Izquierda: parte del usuario; derecha: parte médica.

2.2. Phiby’s Adventures

En el año 2016 se empieza a desarrollar en el grupo GAMMA ejercicios de rehabilitación virtuales diseñados para niños y jóvenes con discapacidad motora. Estos ejercicios se centran en la movilización de brazos y el tronco, de forma que podían realizarse de pie y sentado. Los ejercicios fueron desarrollados de forma que representaban distintas tareas en el entorno virtual: escalar, remar, cortar, volar o bucear. El entorno de desarrollo escogido fue Blender v1 [11] y la captación de movimientos se llevaba a cabo con el dispositivo Kinect Xbox360 (modelo anterior a la Kinect v2).

A partir de este conjunto de ejercicios se diseñó una historia, de forma que los ejercicios quedaban incorporados en ella. Así nace Phiby’s Adventures v1 [12], un juego serio con la finalidad de hacer el ejercicio físico más ameno y divertido para niños y jóvenes con movilidad reducida. Este proyecto trata de Phiby y cómo debe explorar un mapa superando ciertos obstáculos mediante los ejercicios mostrados en la Figura 2 (bucear, cortar madera, remar o trepar árboles) para encontrar a su familia. Este proyecto también fue desarrollado en Blender.

Figura 2. Ejercicios de Phiby's Adventures.

El juego sobre el que trata este trabajo de fin de grado, Phiby's Adventures 3D [13], surge a partir de este proyecto, pero con la idea de crear un mundo abierto y dar al jugador la libertad de explorar el entorno. Para este proyecto se decidió cambiar el motor de juego a Unity 3D por ofrecer más ventajas que el anterior entorno de desarrollo. En este proyecto han colaborado numerosos alumnos de la escuela bien en prácticas o en su proyecto de fin de grado [14, 15, 16, 17, 18, 19, 20, 21, 22]. Además de mantener los ejercicios de la versión anterior, se añaden novedades con el fin de hacer más interesante el videojuego y de animar al jugador a seguir jugando: más personajes, un terreno totalmente nuevo, una historia elaborada y, sobre todo, más interactividad. Los ejercicios están completamente integrados en el transcurso de la historia.

Como se ha explicado anteriormente, este juego hace uso del *middleware* K2UM para poder configurar los parámetros y guardar el progreso en los ejercicios de cada jugador. Se utiliza la Kinect v2 para la captación de los movimientos.

La historia de esta versión trata sobre Phiby, un dragón que vive en la isla Breezeland lejos del resto del mundo con sus cuatro hermanos y su abuela. Sin embargo, un día su hermano mayor Ra decide abandonar la isla. Para eso necesita reunir la suficiente energía para poder escapar y cruzar el océano, y esa energía sólo se encuentra en unos cristales existentes dentro del volcán. Phiby, el más inteligente de los cuatro hermanos, se da cuenta de que, sacando los cristales, el volcán de la isla entraría en erupción y destruiría por completo toda la isla. Phiby y sus otros hermanos no quieren que eso pase, así que intentan detenerle. Pero Ra consigue encerrarles en distintos lugares de la isla. Phiby tendrá liberarse y después encontrar y liberar a cada uno de sus hermanos y así poder pararle los pies a Ra y salvar su hogar. Aquí es donde inicia la aventura el jugador: su objetivo será reunir a todos los hermanos y detener a Ra.

El jugador es quien maneje a Phiby mediante el uso de sus movimientos registrados por la Kinect (echar su tronco hacia delante para andar y correr, y rotarlo hacia los lados para girar). Tendrá que completar una serie de tareas y los ejercicios físicos para ir liberando a sus hermanos, hasta llegar al volcán y detener a Ra. También se puede jugar haciendo uso del teclado y el ratón del ordenador, pero en vez de los ejercicios físicos se mostrará una pantalla simulándolos. Sin embargo, la parte del volcán aún no está desarrollada.

Los ejercicios físicos en esta versión, además del movimiento hacia delante del tronco para realizar el movimiento de andar, son tres:

- El primer ejercicio al que se enfrenta el jugador, representado en la imagen izquierda de la Figura 3, consiste en escalar los barrotes de una jaula en la que se encuentra encerrado Phiby al comienzo del juego. Para el control de la subida, se usan movimientos verticales de los brazos alternando entre uno y otro.
- El segundo ejercicio, representado en la imagen central de la Figura 3, consiste en escalar un árbol para ir recogiendo manzanas. El movimiento es igual que en el primer ejercicio, pero con la diferencia de que en este sí se tiene en cuenta si el movimiento ha sido válido o no según los parámetros establecidos por el terapeuta.
- El tercer ejercicio, representado en la imagen derecha de la Figura 3, se trata de cortar unos troncos. El movimiento requerido es el movimiento vertical, pero de un solo brazo (determinado por el terapeuta), con la particularidad de que el jugador tendrá que mantener unos segundos el brazo en el punto más alto antes de bajarlo.

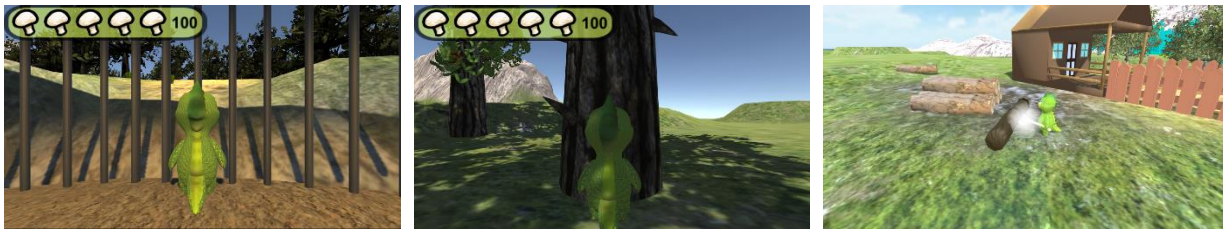

Figura 3. Escenas de los diferentes ejercicios implementados. Izq., primer ejercicio; centro, segundo ejercicio; dcha., tercer ejercicio.

En este proyecto se implementó un sistema de *checkpoints* (también llamados puntos de control) creado por Haoru Quin [19]. Este sistema es bastante frecuente en videojuegos para facilitar el desarrollo de éste y el control de las tareas y ejercicios que ya han sido realizadas por el jugador: los *checkpoints* permiten guardar el progreso del jugador en el juego hasta ese momento.

Phiby’s Adventures 3D se divide en dos capítulos, y cada capítulo consta de varios *checkpoints* que se van superando según se avanza en la historia. El jugador no es consciente de la existencia de estos puntos de control, pues se limita a guardar la partida y cargarla la próxima vez. Son los desarrolladores los que sí hacen uso y cargan la historia en un punto u otro dependiendo de las pruebas que necesiten hacer. Esto les facilita el control de errores y pruebas en el juego, evitándoles tener que empezar desde el principio siempre que quieran comprobar algún cambio o comportamiento.

A continuación, se ofrece una vista rápida de los *checkpoints* implementados:

- *Checkpoint* 0: Escapar de la jaula.

  Se trata del inicio del juego. Phiby se encuentra dentro de una jaula, y deberá escapar. Los barrotes resbalan, así que el jugador deberá usar los montones de paja para poder agarrarse bien. Para poder salir de la jaula, el jugador tendrá que completar el primer ejercicio mencionado anteriormente: escalar los barrotes. Una vez fuera, el *checkpoint* queda guardado.

- *Checkpoint* 1: Llave de la cabaña de la abuela.

  Una vez fuera, Phiby debe ir a la cabaña donde vive su abuela. Por el camino se encuentra señales y animales que le indican a dónde debe dirigirse el jugador. Cuando llega a la cabaña, la abuela de Phiby le cuenta que Ra ha destruido el puente que cruza el río y ha tirado la llave de su cabaña al bosque. El jugador deberá encontrar la llave: puede ser encontrada antes o después de hablar con la abuela.

- *Checkpoint* 2: Recoger manzanas para la abuela.

  Una vez abierta la cabaña, la abuela le pide a Phiby que rellene el bol de dentro con manzanas. El jugador debe dirigirse a los árboles del bosque al lado de la cabaña y completar el segundo ejercicio físico. Si el jugador se queda sin energía, deberá recuperarla descansando y buscando setas, y después volverlo a intentar. Una vez lleno el bol, Phiby devolverá la llave a su abuela, y ésta le dirá que arregle el puente.

- *Checkpoint* 3: Lanzar una cuerda al puente.

  El jugador debe dirigirse al puente. Antes de hacer ninguna interacción en el puente, se deberá de haber encontrado una cuerda en el camino. Una vez recogida, se intentará tirar la cuerda con un movimiento del brazo del jugador, pero Phiby fracasará por no tener suficiente fuerza y decidirá ir a buscar a su hermano Rocky, quien tiene super fuerza.

- *Checkpoint* 4: Liberar a Rocky.

  El jugador deberá dirigirse a las rocas desprendidas en la ladera, al lado del lago. Al acercarse, se oirá a Rocky pedir ayuda y pidiéndole a Phiby que encuentre su superfuerza. El jugador deberá buscar por el bosque cercano y, una vez encontrada y entregada, Rocky se liberará. Rocky decide ayudar a Phiby y le seguirá.

- *Checkpoint* 5: Ir hasta el puente con Rocky.

  Una vez liberado, Rocky seguirá a Phiby hasta el puente. Al contar con la superfuerza de Rocky, esta vez sí se consigue tirar la cuerda, realizando un movimiento del brazo como en el caso anterior.

- *Checkpoint* 6: Después de tirar la cuerda satisfactoriamente.

  Phiby deberá recolectar tablones para poder reconstruir el puente. Para ello, el jugador deberá completar el tercer ejercicio (cortar troncos).

  Una vez obtenidos los tablones necesarios (cantidad determinada por el terapeuta), deberá volver al puente y completarlo. Este *checkpoint* completa el capítulo 1 de la historia.

El capítulo 2 de la historia no ha sido implementado aún, tan solo se ha creado el terreno correspondiente a esa parte.

Es necesario también realizar una pequeña introducción a la estructura de “Phiby’s Adventures 3D” y su entorno de desarrollo Unity. La estructura juego se basa en el uso de escenas de Unity y el uso de scripts para cambiar entre ellas:

- La historia se desarrolla en la escena principal Island. Aquí se encuentran el escenario principal por el que se moverá el jugador, y donde realizará la mayoría de las interacciones (objectos, NPCs (*Non-Player Character*), etc.).
- Cada ejercicio dispone de su propia escena. En ellas, el jugador interactúa para completar el ejercicio y, una vez terminado, vuelven a la escena principal.
- El menú principal también dispone de su propia escena llamada `MenuIntro`.

2.3. Unity, Visual Code y GitHub

Este proyecto está desarrollado con el motor de videojuegos multiplataforma y plataforma de desarrollo de videojuegos Unity [23]. Es de uso libre con la posibilidad de adquirir membresías de pago que dan acceso a funcionalidades exclusivas y de cara a desarrollar contenido con fines comerciales.

La plataforma de Unity permite desarrollar y crear contenido de forma visual y a través de código en lenguaje C++ y C# [24](orientado a objetos) basado en Mono [25] (implementación de código abierto de .NET Framework [26]). Ambos métodos son usados en este proyecto. La versión de Unity usada en este proyecto es la versión LTS_2019.4.21f [27]. Al ser una versión LTS (*long term support*) sufre menos cambios de actualización, beneficiando a proyectos como este que tienen un desarrollo a largo plazo. Además, Unity cuenta con una API excelentemente diseñada y muy accesible, facilitando la información necesaria respecto a sus clases de cara a crear scripts.

Unity ha construido una gran comunidad de usuarios. Gracias a esto se encuentran a la disposición del usuario la posibilidad de obtener *assets* (paquetes de texturas, modelos, animaciones, etc.) creados por otros usuarios de la comunidad a través de la tienda de *assets* de Unity (hay contenido gratuito y de pago). Además, hay gran cantidad de contribuciones de usuarios de funcionalidades desarrolladas mediante scripts. Por último, merece una mención particular todos los paquetes de funcionalidades que se pueden descargar para usar en un proyecto, como puede ser Cinemachine, Recorder (paquetes creados por Unity) o de creación propia por parte de los usuarios.

Para la edición de los scripts se usa el editor de código de libre acceso Visual Studio Code [28] de Microsoft.

Se ha intentado implementar el uso de GitHub [29] en proyectos de Unity para el control de versiones del proyecto. El procedimiento y su uso general son descritos en el Anexo I de este documento para facilitar su aplicación a cualquier proyecto de Unity. Sin embargo, se han experimentado varios problemas, haciendo que el uso de GitHub no haya resultado del todo óptimo. Por ejemplo, al ejecutar escenas directamente desde el proyecto conectado al control de versiones Unity dejaba de responder. También presenta problemas a la hora de ejecutar el proyecto haciendo uso de la Kinect. Es útil para llevar un control de versiones, pero a la hora de trabajar obliga a realizar una copia del proyecto en otra carpeta fuera del repositorio, editar el proyecto y luego copiar esos cambios al repositorio y subirlos.

3. Objetivos del proyecto

Phiby's Adventures 3D es un videojuego funcional que ha sido desarrollado a lo largo de varios años gracias a la contribución de numerosas personas. Es por esto por lo que presenta distintas problemáticas a nivel de diseño de la experiencia del jugador y de la lógica utilizada para implementar sus distintas características.

Este proyecto de fin de grado persigue continuar el desarrollo de "Phiby's Adventures 3D". Lo primero fue dedicar varias semanas a conocer el proyecto y familiarizarse con él, entender el flujo del juego y de su información y determinar las fortalezas y las debilidades del proyecto. De esta forma, se realizó una lista de las posibles partes del videojuego que presentan carencias o posibilidad de mejora o enriquecimiento. Se contemplan tres objetivos principales:

- Mejora del fragmento de historia que trata la liberación del hermano Rocky:
    - Adaptación de la historia original de forma que la experiencia de juego resulte menos tediosa para el jugador.
    - Modificación de las interacciones entre el NPC Rocky y el jugador.
    - Elaboración e incorporación de una nueva escena de ejercicio que transcurra durante dicho fragmento de historia.
- Modernización de la lógica necesaria para añadir cinemáticas dentro del camino que toma el jugador para evitar partes largas que podrían aburrir:
    - Mejora y creación de la lógica necesaria para tener un sistema funcional de reproducción y control de cinemáticas.
    - Perfeccionamiento de la cinemática introductoria ya existente en el proyecto.
    - Creación de dos nuevas cinemáticas con el propósito de mejorar y modernizar la experiencia del jugador.
    - Mejora de la fluidez de los diálogos entre los personajes en estas partes de la historia.

- Actualización del proyecto:
    - Apariencia y diseño del menú principal.
    - Incorporación de sonidos de realimentación al interactuar con coleccionables.
    - Limpieza de archivos obsoletos.
    - Actualización de la lógica usada en los scripts.

Todos estos objetivos se llevarán a cabo de forma que el videojuego siga cumpliendo con todos los requisitos de las versiones anteriores.

4. Mejora de la jugabilidad de *checkpoints* 4 y 5

El primer objetivo de este trabajo de fin de grado es mejorar el fragmento de la historia en la que se trata la liberación de Rocky, uno de los hermanos de Phiby. En este capítulo se trata cómo se ha dado un nuevo enfoque a la historia para arreglar los problemas de jugabilidad que conllevaba, así como la creación e implementación de una nueva escena de ejercicio que forma parte de la nueva narrativa.

4.1. Nuevo enfoque de historia y jugabilidad

Como se introduce en el capítulo 2.2, el desarrollo de la historia está marcado por los *checkpoints*. Entre los *checkpoints* 4 y 5 Phiby debe ir desde el puente hasta la cueva donde está Rocky encerrado porque necesita su ayuda, desde allí debe ir a buscar su super-fuerza al bosque y devolvérsela a Rocky para luego volver con él hasta el puente de nuevo.

Se ha observado que durante el desarrollo de esta parte la jugabilidad se puede volver aburrida. Además, las terapeutas con las que se trabaja en conjunto para el desarrollo de “Phiby’s Adventures 3D” concuerdan en que puede resultar cansado que el jugador realice para andar el mismo movimiento durante mucho tiempo (echar su tronco hacia delante para que Phiby corra o ande). Es por esto por lo que se ha pensado en modificar la historia para dotar a este segmento de una jugabilidad un poco más dinámica evitando que el jugador ante durante mucho tiempo seguido y darle más actividad no solo al jugador sino a la historia.

Lo primero que se ha hecho ha sido modificar la historia en estos dos *checkpoints*, de forma que esta actualización no influye en el resto de la historia ya planteada. La nueva historia ya no requiere que Phiby vaya al bosque a buscar el super-poder, sino que se ha adaptado para que sea él el que libere a Rocky gracias a que éste le presta su super-fuerza desde dentro. Lo segundo ha sido aprovechar esta nueva historia para introducir una nueva escena de ejercicios: el jugador deberá, mediante la realización de movimientos que requieren levantar los brazos de forma horizontal no sobrepasando la línea del hombro, despejar las piedras que cubren la entrada a la cueva donde está Rocky encerrado. Esta escena exige un tipo diferente de movimiento corporal, de forma que se aporta variedad a la jugabilidad en este segmento de la historia.

Además, para darle un nuevo enfoque, se piensa contar parte de la historia mediante el uso de cinemáticas (capítulo 5).

4.2. Implementación de un nuevo ejercicio

En este apartado se explica el proceso de creación de la escena del nuevo ejercicio y la lógica necesaria para realizarlo. Este ejercicio trata de que el jugador vaya levantando sus brazos de forma horizontal, sin sobrepasar su hombro, y de forma alterna. Al realizar estos movimientos, las piedras que cubren la entrada de la cueva donde está Rocky encerrado irán saltando por los aires. La escena de este ejercicio se acaba una vez ocurre uno de los dos siguientes eventos: el jugador completa un número de movimientos mínimos impuestos o se acaba el tiempo para la realización del ejercicio impuestos por el terapeuta desde la web de Blexer.

4.2.1. Creación de la escena

Esta escena se ha diseñado tomando de base una de las escenas de ejercicios anteriores (como puede ser la escena del ejercicio de las manzanas o la de escapar de la jaula) y se ha nombrado como `RockyMinigame`. A continuación, se eliminan aquellos elementos que son prescindibles como el agua, la caseta de la abuela, la jaula, etc., como se puede apreciar en la primera imagen de la Figura 4. Se decide prescindir de estos elementos porque el plano que va a ver el jugador es a Phiby delante de las rocas (segunda imagen de la Figura 4). Una vez se haya completado el ejercicio y, por tanto, despejadas todas las rocas (imagen de debajo de la Figura 4), se verá a Rocky dentro.

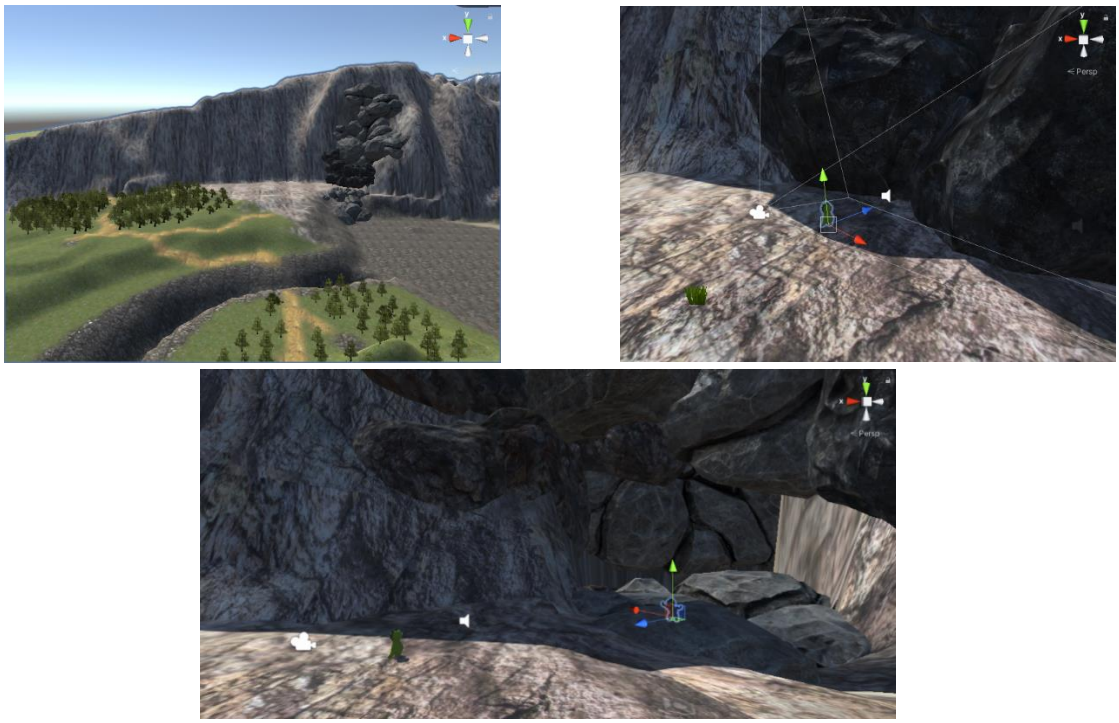

Figura 4. Diferentes ángulos del editor de Unity durante la composición de la escena de ejercicio `RockyMinigame`.

4.2.2. RockMinigameTrigger.cs

Lo primero, una vez creada la escena de ejercicio (capítulo 4.2.1), es implementar la lógica en la escena principal Island para poder efectuar el cambio de la escena de la Isla escena al ejercicio (para el cambio de la escena `MenuIntro` al ejercicio, consultar el proyecto de Juan Fuentes-Pila [1]). En “Phiby’s Adventures 3D” esto se hace mediante el uso de *colliders*. Cada ejercicio dispone de un objeto vacío en la escena `Island` con un *box collider* que actúa de *trigger*, y un script que gestiona el cambio de escena cuando detecta que Phiby ha entrado en contacto con el *collider*. Ejemplos de estos scripts son `ClimbAppleTree.cs` o `ShortBars.cs`, los cuales controlan el acceso a los ejercicios de recoger manzanas y escapar de la jaula respectivamente.

Para esta escena se crea el objeto llamado `TriggerRockElimination` (mostrado en la jerarquía de la escena `Island` en la primera imagen de la Figura 5), situado cerca de las rocas que tapan la cueva donde se encuentra Rocky. Este objeto, además de contener el *collider* (mostrado en la imagen derecha de la Figura 5), contiene el script `RockMinigameTrigger.cs`, creado para encargarse de hacer el cambio entre escenas. Se debe destacar que este objeto se encuentra inicialmente desactivado debido a que antes de cargar la escena del ejercicio se debe visualizar una cinemática (se explica más en el capítulo 5). Cuando se termina la reproducción de la cinemática, se activa el objeto y ya se produce la interacción con el *collider*.

Figura 5. Objeto `TriggerRockElimination` en el editor de Unity.

`RockMinigameTrigger.cs` comprueba primero si el jugador dispone de suficiente energía para intentar completar el ejercicio. esto se hace comprobado si valor de la variable `onScreenEnergy` almacenada en el script `GameState.cs`) es mayor que un valor determinado. Si al jugador le falta energía, se reproducirá un audio en el que Phiby comenta que está cansado y necesita reponer energía. Si, por el contrario, cumple el requisito de energía, el script cambia la variable `gameType` del `GameState` que controla qué ejercicio se está ejecutando y va a comprobar si el jugador está jugando con teclado o con la Kinect:

- Con teclado: se carga la escena `MiniGamesSimulator`. Esta escena se trata de un objeto Canvas (área en la que se deben colocar todos los objetos pertenecientes a la interfaz de usuario, que deben ser hijos de este objeto), y muestra cierta información sobre el ejercicio en vez del ejercicio en si (descripción y qué hacer para terminar la simulación del ejercicio). La información mostrada depende de la variable `gameType`.
- Con Kinect: se carga la escena `RockyMinigame`, pasando también el valor de la variable `gameType`.

Como el jugador está jugando con la Kinect, se entra en la escena `RockMinigame`. Se va a pasar a explicar el flujo de los dos scripts que intervienen en el control y la realización del ejercicio: `RockMinigame.cs` y `MovementControl.cs`.

4.2.3. RockMinigame.cs

Una vez se tiene la forma de entrar a la escena del ejercicio implementada, se debe implementar un método que permita controlar y cuantificar los movimientos que realiza Phiby. Para ello, se ha adaptado el método que ya utilizaban las escenas de los ejercicios de coger manzanas y escapar de la caja. Este método consiste en el uso de dos esferas *collider* con función de *trigger* situadas al alcance de las manos del modelo de Phiby. En ejercicios anteriores estas esferas estaban situadas por encima de la cabeza de Phiby ya que el movimiento consistía en levantar los brazos. Para este nuevo ejercicio se quiere que el jugador levante los manos de forma horizontal y alterna, así que las esferas con *collider* se sitúan a ambos lados de Phiby y dentro del rango de sus brazos.

`RockMinigame.cs` es el script que controla y reconoce los movimientos que realiza el jugador, y está basado en los scripts usados en los ejercicios de escapar de la jaula y recoger manzanas (`ClimbCage.cs` y `ClimbingTrees.cs` respectivamente). La forma que tiene este script de detectar un movimiento del jugador es ayudándose de esas dos esferas con *collider* (mostradas en la primera imagen de la Figura 6); cada una de las esferas tiene como componente el script `RockMinigame.cs`. Este script reconoce cada vez que uno de los brazos de Phiby entre en contacto con el *collider* de esa esfera y registrará ese movimiento.

Figura 6. Izq.: Escena `RockyMinigame` en el editor de Unity mostrando los *colliders* esféricos; dcha.: misma escena desde la vista de juego.

Este scripts se desarrolla como sigue: primero, en el método `Update()`, mientras que no se indique mediante la variable `movementRegistered` que el jugador ya está realizando un movimiento, se esperará a detectar que la coordenada local *y* de una mano del esqueleto (`IKTargetR` o `IKTargetL`, se debe asignar mediante el inspector) sea mayor que cero, indicando que se ha realizado un movimiento de la mano respecto de la posición en reposo y restando la energía. Si el movimiento realizado hace que la mano de Phiby entre en contacto con el *collider* de su esfera (detectado mediante la función `OnTriggerEnter()`), se considera que el movimiento ha sido correcto; si no, el movimiento será calificado de incorrecto (ambos tipos de movimiento reciben realimentación sonora). Una vez se ha detectado que la coordenada local *y* de la mano levantada vuelve a ser menor que cero, se desactiva el *bool* `movementRegistered` para que se pueda captar el siguiente movimiento.

Cuando se termina la implementación total de éste y el resto de los scripts y se prueba el ejercicio, se encuentra con el problema de que el modelo de Phiby no responde bien al emparejamiento del esqueleto de la Kinect. Como se puede observar en la segunda imagen de la Figura 6, sus brazos están apuntando hacia delante en vez de en posición de reposo a ambos lados de Phiby. Al ejecutar el ejercicio cuenta todos los movimientos como erróneos. No ha habido tiempo para poder implementar ninguna solución, por lo que se incluye en el capítulo 10 como futuro trabajo.

4.2.4. MovementManager.cs

`MovementManager.cs` es el script que controla que se cumplan los parámetros establecidos por el terapeuta (descargados previamente a través del K2UM) durante la realización del ejercicio, y se encuentra en el objeto vacío `EventMng` de cada escena. En versiones anteriores se llamaba `ClimbMovementManager.cs`, pero al introducir un nuevo ejercicio que no implica escalar, se ha decidido cambiarle el nombre a uno más representativo. Lo primero que hace este script nada más se cambia a la escena del ejercicio es cargar los datos almacenados en el script `WebData.cs` que correspondan al ejercicio que se esté ejecutando, indicado por la variable `gameType` del script `GameState.cs`. Mientras se está realizando el ejercicio, va a comprobar si se acaba el tiempo o si se han cumplido los parámetros (ambas variables establecidas por el terapeuta). Cuando se cumple una de estas condiciones, se activa la variable `exerciseEnd` del script `KinectGamesManager.cs` (script que controla la ejecución de los ejercicios en modo Kinect y el simulador de ejercicios), indicando que la ejecución del ejercicio se ha terminado y se debe volver a la escena principal Island. El funcionamiento y trabajo de este script no ha sido modificado sino actualizado debido a la actualización de los scripts de comunicación con la Kinect (como se describe en el capítulo 8.2).

Sí se ha añadido en el script `MovementManager.cs` el control de las animaciones para poder animar que el jugador va quitando rocas a medida que realiza movimientos (estas rocas están agrupadas bajo el objeto vacío `RockyTrap`). En Unity, un objeto puede tener varias animaciones, y el control de animaciones en Unity se basa en el uso de estados de animación entre los que se va cambiando, dependiendo del valor de unas condiciones o variables determinadas; para este caso se ha decidido que las condiciones sean de tipo *bool*.

Para controlar este se hace uso del componente `Animator`. Para acceder a estas variables y cambiar su valor cuando se considere necesario mediante el código se debe indicar al script, a través el inspector, los componentes `Animator` de cada una de las cinco rocas. En la Figura 7 se puede observar un ejemplo del `Animator` de una de las rocas.

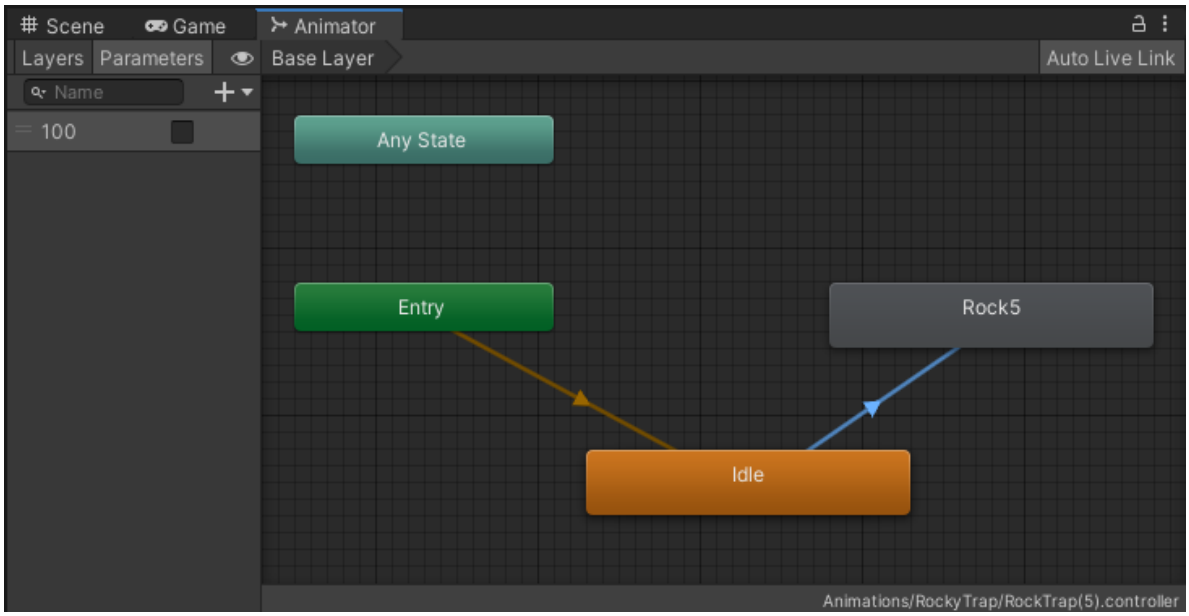

Figura 7. Ventana del `Animator` de una roca, con sus estados de animación y la variable que permite transicionar entre ambos estados.

La forma empleada para calcular en qué momento se debe activar la variable correspondiente a la animación de cada roca (es decir, reproducir su animación) es la siguiente: las piedras irán saltando por los aires de forma ordenada cada determinado número de movimientos realizados por el jugador. Al haber cinco rocas animadas, se ha pensado que se reproduzca cada animación según se vaya completando un porcentaje de los movimientos requeridos por el terapeuta: 20%, 40%, 60%, 80% y 100%. Por ejemplo, si se requieren diez movimientos para completar el ejercicio, la animación de la primera roca se reproducirá cuando se hayan completado dos movimientos; la segunda cuando se hayan completado cuatro movimientos, y así sucesivamente hasta que al décimo movimiento se haya quitado la última roca. En la reproducción de la última animación siempre se esperan unos segundos para que dé tiempo a que la animación se reproduzca antes de que se cambie de escena.

Si se desea controlar animaciones específicas para una escena de ejercicio, es importante incluir ese código dentro del `if()` que distingue cada ejercicio por su `gameType`.

4.3. Nuevo control del movimiento de Rocky

En la anterior versión, una vez liberado, Rocky agradece a Phiby su ayuda, y el jugador debe volver al puente con Rocky a su lado para que le pueda prestar su super fuerza. Para esto, Haoru Quin [19] programó para Rocky un seguimiento automático hacia Phiby por medio de una malla de navegación (`NavMesh`). Sin embargo, Rocky se queda atascado en algunas partes del terreno y en algunas ocasiones se desliza por algunas pendientes. Después de que varios alumnos

intentaran corregir estos fallos sin éxito, y después de comprobar que la malla de navegación del terreno (determina por qué partes del terreno pueden avanzar los objetos con un agente de navegación) estuviera bien generada, se decide junto con la directora del proyecto en pensar en posibles alternativas para que Rocky vaya al puente junto con Phiby.

La mejor decisión encontrada es la de cambiar los roles y que sea Phiby quien sigue a Rocky hasta el puente. Es decir, diseñar un camino predeterminado por ciertos puntos por el que Rocky vaya automáticamente y sin quedarse atascado ni resbalarse, y que sea el jugador que deba mover a Phiby de manera que le siga. Para que no haya problemas y Rocky no se marche sin que el jugador tenga oportunidad de andar con él, se ha programado el en script `RockyController.cs` que compruebe cada vez que Rocky llegue a uno de los puntos del camino si Phiby se encuentra dentro de un radio determinado. Si Phiby se encuentra dentro de ese radio, Rocky continúa andando sin detenerse; en cambio, si no se detecta a Phiby en ese radio Rocky se para en el siguiente punto al que se esté dirigiendo y espera a que Phiby esté cerca. De esta forma, el jugador no se ve agobiado por tener que seguir cerca a Rocky todo el tiempo ni por perderse, y si fuera necesario podría ir a buscar alguna seta si necesitase reponer energía. Además, la velocidad de Rocky también es adaptable por el desarrollador. Se ha optado por dotar a Rocky de un poco más de velocidad que Phiby pues se prefiere que espere él y no el jugador. El funcionamiento de la lógica y los scripts empleados para el control de Rocky se explican de forma detallada en el capítulo 4.3.1. A continuación, se explica la implementación de este comportamiento dentro de Unity y el código necesario.

El camino que seguirá Rocky se crea mediante la colocación de 10 objetos vacíos en la escena (inicialmente se han usado 10 puntos, pero si se quisiera hacer un camino con más puntos simplemente sería añadir más objetos vacíos). La posición de esos objetos (ordenados para seguir la ruta) serán las coordenadas a las que se dirigirá Rocky cuando sea oportuno, y se muestra su localización en la Figura 8.

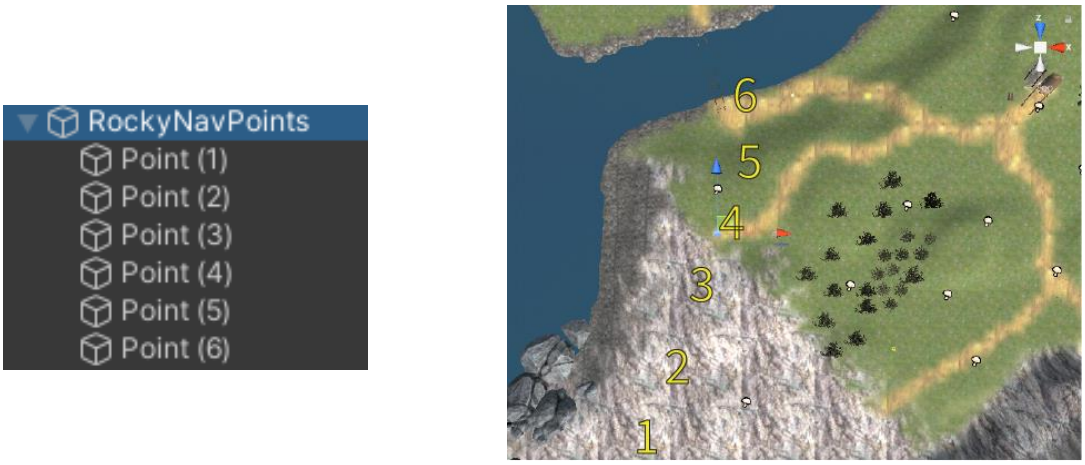

Figura 8. Puntos que constituyan la ruta de Rocky.

4.3.1. Implementación de un sistema de navegación por puntos

Unity dispone de un sistema de navegación, el cual permite que los personajes (u objetos) de un juego puedan moverse por ese mundo (incluso si hay escaleras, saltos, etc.) [30]. Para que este sistema funcione, hacen falta los dos siguientes componentes:

- `NavMesh`: llamada malla de navegación. Son los datos que describen sobre qué superficies se puede caminar. Estos datos necesitan ser construidos sobre la geometría de cada nivel.
- `NavMesh Agent`: es un componente que se debe añadir al objeto que se quiere que se mueva por el nivel. Permite que el objeto navegue utilizando los datos de la malla de navegación y evite los obstáculos. Se representan mediante cilindros.

Para crear una `NavhMesh` [31] se deben seguir los siguientes pasos:

1. Seleccionar la geometría del nivel que queremos que sea apta para caminar.
2. Abrir la ventana de navegación (*Window* > *AI* > *Navigation*) y en la pestaña `Object` marcar la casilla `Navigation static` para que se incluyan esos objetos en el cálculo de la `NavMesh` como muestra la Figura 9.
3. Ajustar en la pestaña `Bake` los parámetros acordes al tamaño del objeto (el agente) que va a caminar sobre la malla según muestra la Figura 10.
    a. `Agent Radius`: controla cómo de cerca el centro del agente puede estar de paredes o bordes.
    b. `Agent Height`: controla cómo de bajos pueden ser los espacios para que el agente los alcance.
    c. `Max Slope`: decide la máxima inclinación de las rampas que puede navegar el agente.
    d. `Step Height`: define el escalón máximo en las cuales el agente se puede parar.
4. En la misma pestaña anterior, hacer clic sobre el botón `Bake` para calcular la malla de navegación.

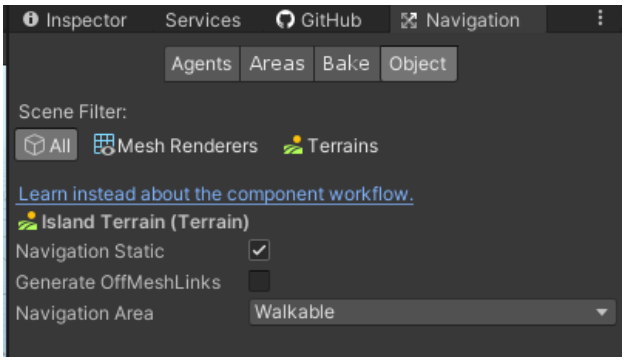

Figura 9. Ventana de Navegación para marcar el terreno para calcular la malla de navegación (paso 2).

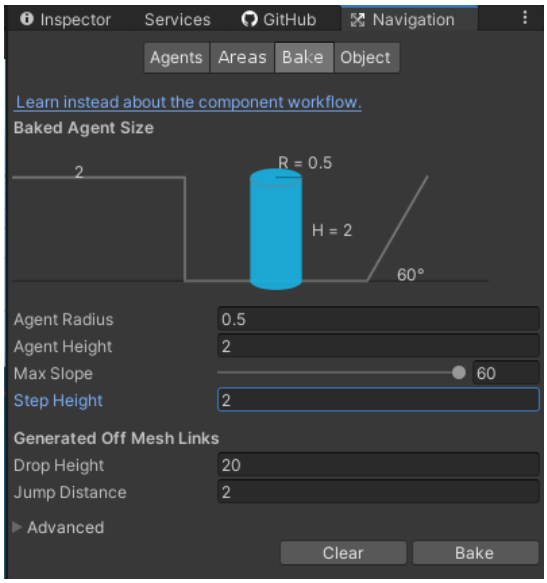

Figura 10.Parámetros para el cálculo de la malla de navegación (pasos 3 y 4).

Mientras la ventana de navegación esté activa, la malla de navegación se muestra como una superposición azul en el terreno del nivel, según se muestra en la Figura 11.

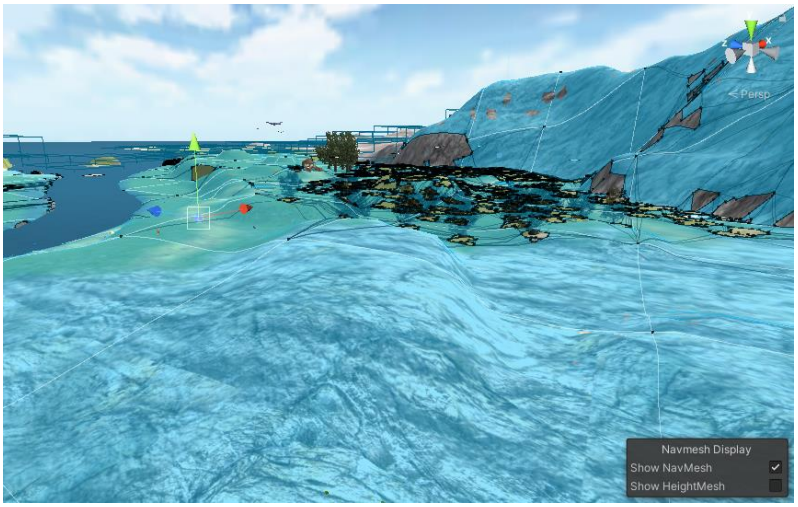

Figura 11. Malla de navegación ya calculada.

Una vez obtenida la malla de navegación, se debe añadir el componente `NavMesh Agent` [32] al objeto que se quiera que navegue por el terreno. En este caso, será el objeto del personaje Rocky.

Para configurar este componente, primero se debe elegir un tipo de agente. En este caso, se ha creado uno para Rocky con las características que se muestran en la Figura 12. El resto de las opciones se pueden modificar libremente pues controlan distintas características del desplazamiento del objeto. Las más importantes son las siguientes:

- `Speed`: velocidad máxima de movimiento del objeto.
- `Angular speed`: velocidad de máxima de rotación.
- `Acceleration`: aceleración máxima.
- `Stopping distance`: distancia a la que para el agente cuando esté cerca de su ubicación destino.
- `Auto braking`: si está activada, el agente frena automáticamente al acercarse al destino.

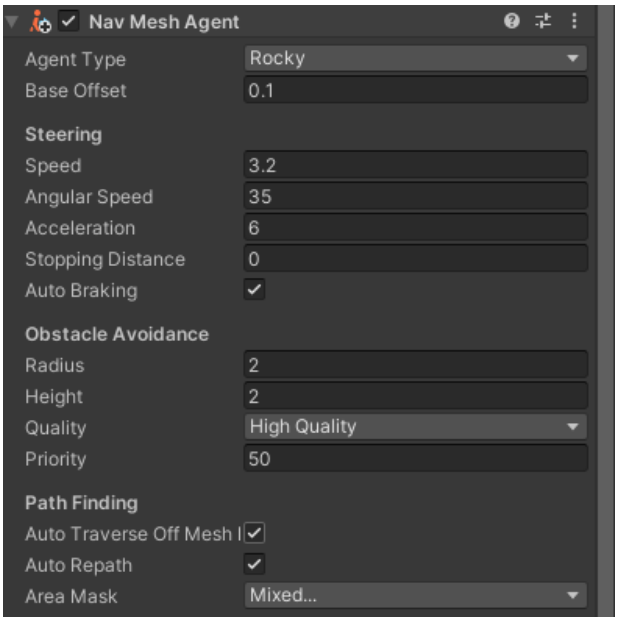

Figura 12. Parámetros configurables del componente `NavMeshAgent`.

El siguiente paso es redactar el código que dirige a Rocky a los puntos colocados en el mapa. Este código se incluye en el script ya existente `RockyController.cs` para evitar crear demasiados scripts y aprovechando que en este script ya se trataba el movimiento anterior de Rocky (seguir a Phiby) (más información sobre este script en el capítulo 4.3.3). Primero se deben definir las variables necesarias:

- `Transform[] points`: los puntos de la trayectoria de Rocky (los objetos vacíos creados al inicio de este capítulo 4.3) se guardan en un array de objetos. Deben introducirse en el array en el orden de llegada (de principio a fin) mediante el inspector.
- `float radio`: variable para ajustar el radio para detectar a Phiby.
- `int destPoint`: inicializada a cero para indicar el primer elemento del array de puntos.

Las variables que detectan la malla de navegación y a Phiby ya existían: `nav` y `phibyTransform` respectivamente.

El código que controla el movimiento se coloca en el método `Update()` del script pues se debe estar controlando todo el rato a dónde debe dirigirse Rocky. Primero comprueba que el jugador se encuentre en el *checkpoint* 5 y la variable `power` sea cierta, lo cual ocurre cuando Rocky le da las gracias a Phiby después de haberle liberado. Una vez terminada esa conversación,

se calcula si Phiby se encuentra a una distancia menor que el radio de detección de Rocky. En este caso, si la distancia al siguiente punto del camino es menor a una cierta cantidad y si no hay ningún camino siendo preparado aún (esta última condición se debe a que mientras el `NavMesh Agent` está en movimiento, está recalculando su ruta y sólo cuando éste llega a su destino se quiere que empiece a calcular su siguiente ruta), Rocky se dirige hacia el siguiente punto del array. Al empezar el camino siempre va a ser verdadero pues para hablar con Rocky el jugador necesita estar a su lado. Este algoritmo se repetirá hasta que se alcance el último punto y se cambie al *checkpoint* 6.

Para poder actualizar el destino de Rocky al siguiente punto una vez ha alcanzado el actual, se ha implementado el método llamado `GotoNextPoint()`. Este método se llama cada vez que se actualiza el siguiente punto al que debe dirigirse Rocky y se basa en la asignación del valor de la posición del array de puntos a la propiedad del componente `NavMesh Agent destination`. Se va recorriendo el array según se van alcanzando los distintos puntos.

4.3.2. Implementación de audios durante una navegación por puntos

Para que al jugador no le resulte tan aburrido seguir a Rocky, se ha implementado un sistema que reproduce audios aleatoriamente cuando se llega a uno de los puntos que marca el camino de Rocky.

Para ello se deben añadir componentes de emisión de sonido y *colliders* en cada punto de la ruta de Rocky como muestra la figura Figura 13, y, cada vez que Phiby entre en el *collider*, se decidirá de forma aleatoria si se escucha un audio de Rocky o no.

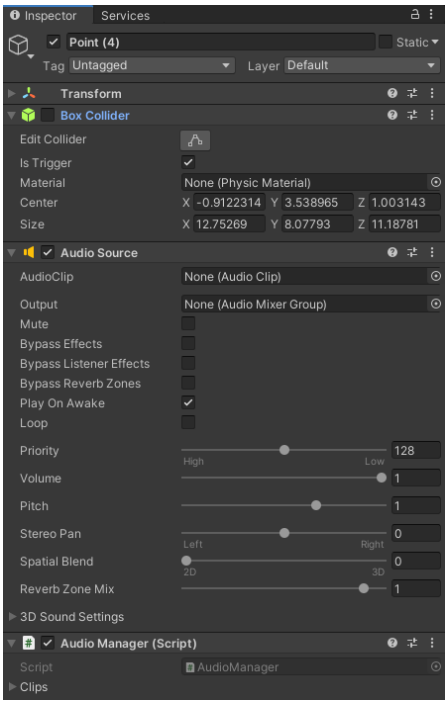

Figura 13. Ejemplo de los componentes de un punto en el que se quiera que haya posibilidad de reproducir un audio de Rocky.

Todo esto está controlado por el script `RockyAudioController.cs`. Este script se debe de añadir como componente a cada punto de la ruta en el que se quiera poder escuchar a Rocky. Cada vez que detecte que Phiby ha entrado en contacto con el *collider*, primero decidirá, de forma aleatoria, si en ese punto se escucha un audio o no. En el caso de que sí se escuche, reproducirá un audio aleatorio del array de clips de audio.

En esta versión se han añadido *colliders* a los puntos número dos, cuatro y ocho, pero en el futuro se puede modificar a decisión de los desarrolladores. Estos *colliders* deben estar inicialmente desactivados. El código se encargará de activarlos solamente cuando el jugador se encuentre en el *checkpoint* 5.

En total se han grabado cinco audios para este *checkpoint*. Juan Fuentes-Pila es quien presta su voz para las líneas de Rocky.

4.3.3. Explicación del script RockyController.cs

Este script recoge todas las interacciones que tiene Rocky con el entorno y con Phiby. Debido a las modificaciones que han sido realizadas en su historia y la mecánica de movimiento de Rocky, ha sufrido bastantes alteraciones y se cree necesario explicar su funcionamiento para el futuro desarrollo del proyecto.

Para empezar, se han eliminado todas las variables que tenían que ver con la antigua misión de buscar la super-fuerza de Rocky en el bosque, y, por tanto, todo el código que hacía funcionar su lógica pues ya no se van a usar en ningún momento de la historia:

- `SuperPower superPower`
- `GameObject superPowerContainer`
- `Bool superPowerRocky`

No obstante, sí se mantiene la variable que controla el comportamiento del halo que obtiene Phiby al estar cerca de Rocky pues esta característica no sufre alteraciones con el cambio de historia.

Después del método `Start()` (el cual inicializa todas las variables y obtiene el resto de variables u objetos que no se han obtenido mediante el inspector de Unity) se encuentra el método que controla la reproducción de audios dependiendo del *checkpoint* en el que se encuentre el jugador mediante el uso de un `switch`. Aquí se han eliminado dos audios pertenecientes al *checkpoint* 5 porque ya se utilizan en la cinemática que se reproduce al principio de este *chekpoint*. También se ha quitado el `case default` pues al haber insertado audios en el camino no es necesario reproducir un audio cada vez que Phiby entra en el trigger de Rocky. Los siguientes cambios que se hicieron al script ha sido añadir el método `GotoNextPoint()` y el control de movimiento de Rocky en el método `Update()`. Para acabar, se tienen dos métodos, uno que controla cuándo Phiby permanece dentro del *collider* de Rocky (`OnTriggerStay()`) y otro que controla cuándo Phiby deja de estar en contacto con ese *collider* (`OnTriggerExit()`). El primero controla que el halo que simboliza la super-fuerza aparezca sólo cuando Phiby se encuentra dentro del *collider*; el segundo controla que el componente del halo se desactive, así como que se pare la reproducción de audio cuando Phiby se aleja de Rocky y sale de su *collider*.

5. Cinemáticas

Las cinemáticas (o escenas) son secuencias de video reproducidas en un cierto momento durante el desarrollo de la experiencia de juego. Normalmente el jugador no tiene o tiene muy poco control sobre el desarrollo de estas e interrumpen el juego.

Con el fin de enriquecer la experiencia del jugador, las cinemáticas son usadas con múltiples fines:

- Avance de la trama de la historia del videojuego.
- Presentación de NPCs, enemigos, etc.
- Facilitar información (atmósfera, diálogos, pistas, escenario, etc.).
- Eventos clave para el protagonista (fortalecimiento, nueva equipación, etc.).

Según el método de producción usado, las cinemáticas de un videojuego pueden clasificarse de la siguiente manera:

- De imagen real: una grabación que hace uso de escenografías reales y actores para la representación de los personajes.
- Animadas: usan animación 2D o 3D.
    - Pre-renderizadas (FMV (*Full Motion Video*)): una grabación de una secuencia de animación.
    - Renderizadas en tiempo real: la secuencia se renderiza con el mismo motor del juego.
    - Mixtas: una combinación de las dos técnicas anteriores.
- Interactivas: suelen hacer uso de los eventos de tiempo rápido (más conocidos en la industria por su nombre de habla inglesa *quick time events*). La máquina toma el lugar del personaje que maneja el jugador mientras van apareciendo distintas indicaciones (como pulsar botones, mover un *joystick*) que tiene que completar el jugador para tener éxito.

5.1. Diseño de las cinemáticas

Phiby’s Adventure 3D es un juego pensado para tener una alta interacción con el jugador mediante el uso de la Kinect, pudiendo incluso llegar a no dejar mucho descanso al jugador.

La introducción de cinemáticas nuevas pretende poder dar un pequeño descanso al jugador del movimiento físico que realiza para controlar al personaje principal a la vez que agiliza el proceso de narración de la historia del videojuego. Es por eso por lo que para llevar a cabo esta tarea se opta por el uso cinemáticas animadas no interactivas.

Respecto a la producción de dichas animaciones se decide usar escenas pre-renderizadas. La principal razón detrás de esta decisión es la de evitar la carga de computación que añade el renderizado en tiempo real al ordenador. Esto permite mantener los requisitos de hardware para jugar al videojuego más al alcance de cualquier ordenador de gama media - baja.

Por último, se decide mantener el botón de ***skip*** que se encuentra en la escena Intro del juego y añadirlo para que aparezca siempre que haya una cinemática.

5.1.1. Reestructuración de la cinemática de introducción

La cinemática existente en el proyecto se reproducía siempre que se iniciaba el juego, y se trataba de dos planos.

Figura 14. Planos de la cinemática de introducción al juego original.

El primer plano que muestra un vuelo sobre la isla (escenario del videojuego). Fue grabado con una sola cámara. Se puede ver uno de los fotogramas en la imagen izquierda de la Figura 14. En el segundo plano se ve a Phiby dentro de la jaula en la que está encerrado al empezar la historia, recitando un monólogo en el que introduce al jugador en el mundo y en la historia del videojuego. Se incluyen subtítulos, y se muestra un fotograma en la imagen derecha de la Figura 14. Esta cinemática sirve para introducir al jugador en el mundo y en la historia de forma que se puedan sentir inmersos desde el principio. Contenía también música de fondo, así como diálogo que cumplían las funciones anteriores.

Sin embargo, salvo que sea la primera vez que el jugador ingresa en el juego, puede resultar tedioso oír a Phiby todas las veces explicarnos la historia. Además, la aparición de los subtítulos no es muy adecuada y debido a la calidad del video (el video tiene pocos *frames* por segundo) puede resultar poco legible. Es por estas razones que se decide dividir esos dos planos en dos cinemáticas separadas.

La primera de ellas se seguirá reproduciendo al iniciar el juego para contribuir a la inmersión del jugador en el mundo. Se ha cambiado un poco el guión, pues la cámara no se dirige hacia a jaula sino hacia la cabaña, donde estará Phiby esperando como muestra la imagen izquierda de la Figura 15. De esta forma, la presentación es general. La segunda cinemática se reproducirá cuando se empiece una nueva partida, permitiendo al nuevo jugador conocer ese contexto necesario para no desmotivarse en el juego. Además, se sustituyen los subtítulos anteriores por unos que muestran menos texto a la vez, y se van actualizando según el diálogo de Phiby como muestra la imagen derecha de la Figura 15.

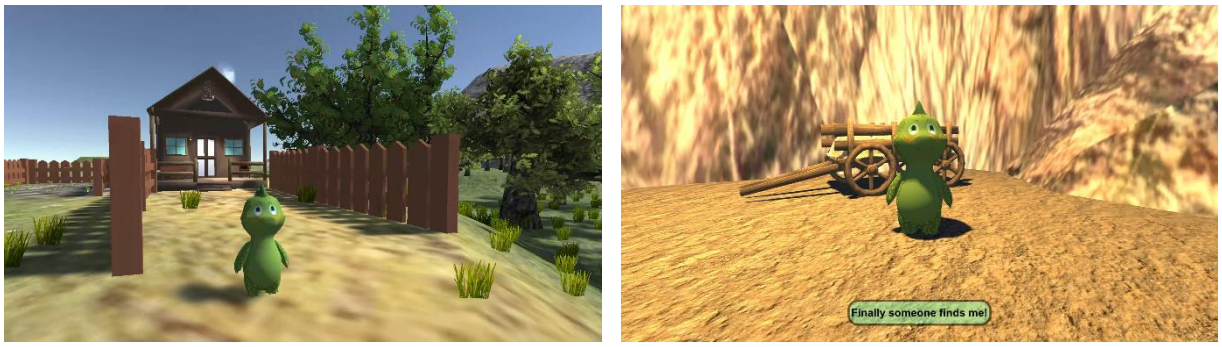

Figura 15. Planos de las nuevas cinemáticas: primera (izquierda) y segunda (derecha).

5.1.2. Cinemáticas nuevas de *checkpoints* 4 y 5

Con la modificación de la historia y, por tanto, de la estructura de los *checkpoints* 4 y 5, se decide hacer uso de las cinemáticas con el fin de hacer menos tedioso el camino hasta la cueva de Rocky y para hacer más fluidos los diálogos entre Phiby y Rocky en vez de depender de si están cerca o se repiten los audios (ya que en varias ocasiones se solapaban los audios o repetían).

Para crear una cinemática nueva, primero se diseña un guión gráfico (*storyboard*) para previsualizar los planos que se quieren mostrar en cada secuencia. Se usan diferentes ángulos de cámara y seguimiento a los personajes para dar más dinamismo a la secuencia. En la Figura 16 se pueden observar un guión gráficos diseñado para ambas cinemáticas.

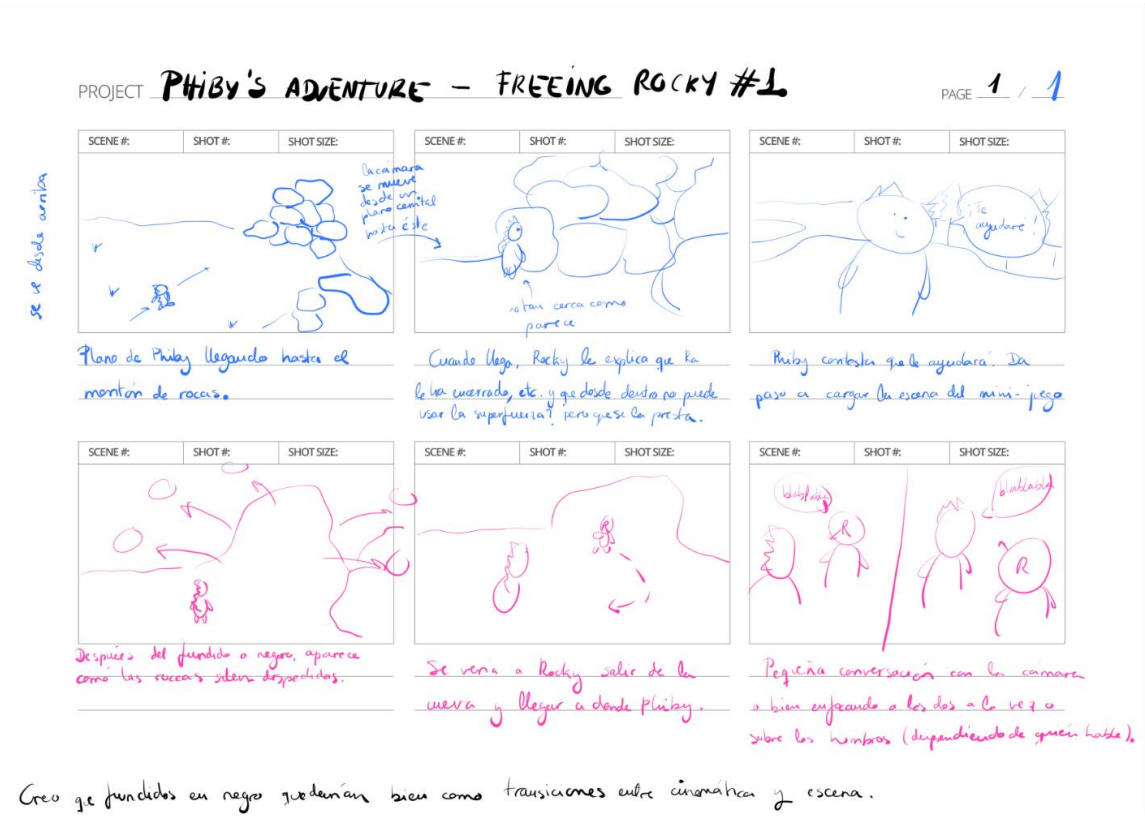

Figura 16. *Storyboard* de las nuevas cinemáticas: arriba, *checkpoint* 4; abajo, *checkpoint* 5.

La cinemática del *checkpoint* 4 se activa la primera vez que el jugador, estando en ese *checkpoint*, se acerca a una cierta distancia de la cueva. En ella se puede ver a Phiby corriendo hacia las rocas (ahorrando al jugador el recorrido tedioso). Una vez cerca, Rocky le pedirá que le libere. Las dos imágenes superiores de la Figura 17 muestran dos fotogramas de esta cinemática.

La cinemática del *checkpoint* 5 (reproducida también una sola vez inmediatamente después de terminar el ejercicio de liberar a Rocky) se puede ver desde lejos cómo las piedras salen despedidas y Rocky sale de la cueva. Rocky da las gracias a Phiby y le dice que le ayudará a reconstruir el puente. Las dos imágenes inferiores de la Figura 17 muestran dos fotogramas de esta escena.

Los diálogos ya existentes son coherentes, pero se decide regrabar las líneas de Rocky (con ligeras modificaciones) para dotarlas de más carácter. Es Juan Fuentes-Pila quien presta su voz para estas escenas.

Figura 17. Planos de las nuevas cinemáticas: arriba, *checkpoint* 4; abajo, *checkpoint* 5.

5.2. Proceso de grabación de una cinemática en Unity

La cinemática ya existente se produjo grabando la pantalla mientras se ejecutaba el juego desde una escena determinada (una que no existe en el proyecto, debió de ser eliminada). Los elementos de esas escenas constaban del terreno de la isla, el título del videojuego, los subtítulos y la música y audios de Phiby.

Este sistema de creación y grabación de escenas y cinemáticas ha sido mejorado sustancialmente de forma que futuros desarrolladores sean capaces de crear cinemáticas en la isla desde un mismo lugar o regrabar las que ya existían.

Para empezar, se ha creado una nueva escena dedicada exclusivamente a la grabación de todas las cinemáticas que se pudieran querer incluir en el videojuego y que transcurran en el

terreno de la isla. Esta escena, llamada Cinematics, se ha creado a partir de la escena principal Island, dejando solo el terreno de la isla (incluida la jaula y la cabaña) y los animales y deshabilitando el resto de los scripts y *triggers*. De esta forma se pueden incluir los elementos necesarios y activar/desactivar aquellos que no vayan a ser de uso mientras se graba la escena correspondiente. La estructura de esta escena se puede apreciar en la Figura 18, donde por defecto están desactivadas los elementos de las cinemáticas ya creadas.

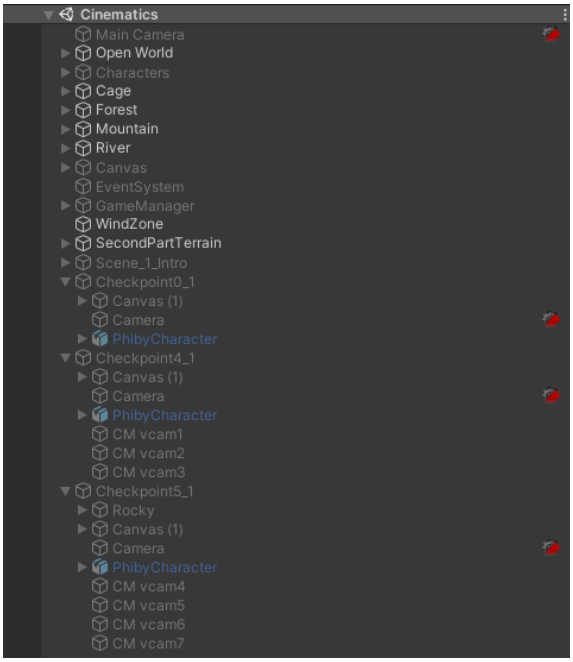

Figura 18. Jerarquía de la escena de montaje y grabación de cinemáticas.

El tener una escena dedicada enteramente al montaje y creación de cinemáticas permite que, si fuera necesario regrabar alguna cinemática o realizar algún pequeño ajuste, no sea necesario realizar de nuevo todo el montaje de la escena, animaciones, etc. Tan sólo sería necesario modificar esta escena si la escena principal Island sufriera una modificación que influenciara cinemáticas ya existentes o planeadas.

En el caso de querer crear una nueva cinemática, se debe crear primero un objeto vacío y nombrarlo de forma que sea distinguible de otras cinemáticas. Dentro de ese objeto vacío se deber mover o copiar todos los objetos (personajes, elementos del terreno, canvas para subtítulos, cámaras, etc.) que van a aparecer en la cinemática. De esta forma con solo activar o desactivar ese objeto se activaría o desactivaría el resto de los elementos.

Por ejemplo, en la Figura 19 se muestran los elementos usados para montar las cinemáticas de los *checkpoints* 4 y 5, que son las que más elementos han requerido (el modelo de Phiby y de Rocky, un objeto `Canvas`, y las cámaras). Se debe destacar que para grabar la cinemática del *checkpoint* 5 es necesario que el componente `Animator` del objeto `RockyTrap` esté activado para que se pueda reproducir la animación de explosión de las rocas.

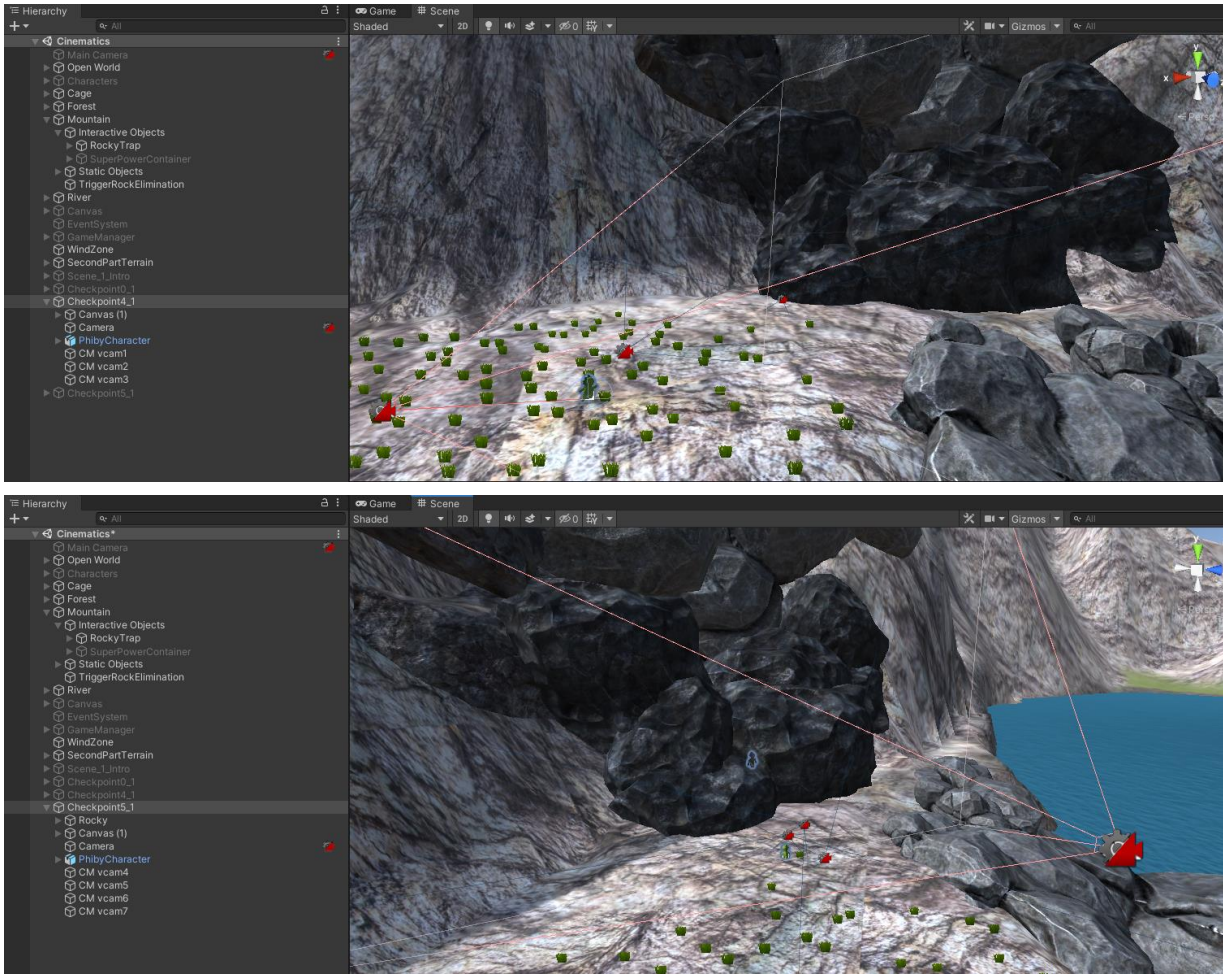

Figura 19. Elementos usados en el montaje y grabación de las cinemáticas para los *checkpoints* 4 y 5.

Una vez reunidos todos los elementos que se quieren usar en la cinemática (la incorporación de las cámaras virtuales se explica en el apartado 5.2.2), es momento de colocarlos en el tiempo y el espacio.

A continuación, se ofrece una explicación del método seguido para montar y grabar una cinemática, que puede tomarse como referencia para futuros trabajos.

5.2.1. Montaje de los elementos de la escena de la cinemática con Timeline

Para colocar los elementos que se quiera que aparezcan en la cinemática se va a usar el paquete `Timeline` [33]. `Timeline` permite la creación de contenido cinematográfico, cinemáticas, etc., de una forma muy sencilla. Cada escena que se crea con la línea de tiempo de Unity se compone de una línea de tiempo (*timeline*) y una instancia de un *asset* de `Timeline`.

Para empezar, se debe abrir la pestaña de `Timeline`, se debe seguir la ruta *Window* > *Sequencing* > *Timeline*. Se selecciona el objeto deseado en el que se quiere crear la línea del tiempo y se debe crear un componente de director y un *asset* de `Timeline` según muestra la Figura 20.

Aparece una interfaz que es bastante parecida a un software de edición de video estándar. Es recomendable que esta línea del tiempo se cree sobre un objeto vacío que no se vaya a usar para nada más en la cinemática, y que esté nombrado de forma que represente la cinemática.

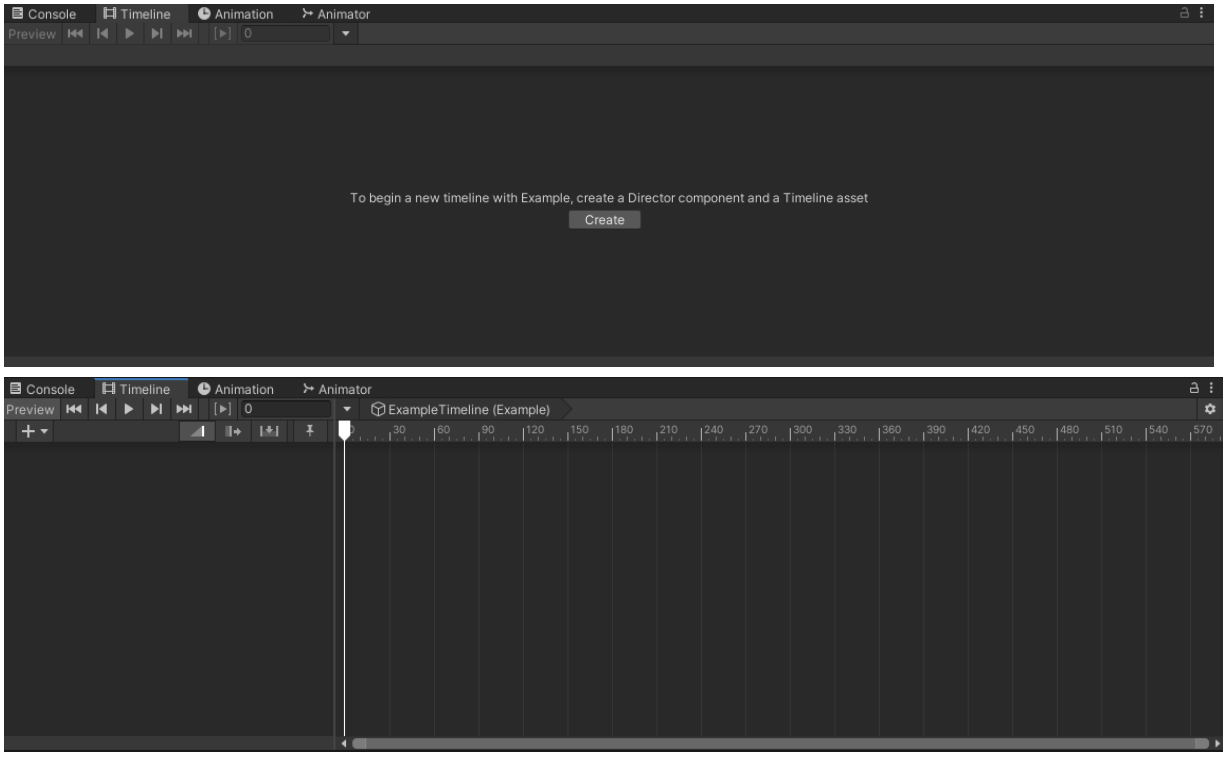

Figura 20. Creación de una línea de tiempo en `Timeline` sobre un objeto.

En el proyecto se han creado los objetos vacíos marcados en azul en la Figura 21, que tienen como hijos al resto de objetos usados en las correspondientes cinemáticas.

Una vez obtenida la línea del tiempo en el objeto vacío como la mostrada en la imagen inferior de la Figura 20, es hora de empezar a montar los distintos elementos. A continuación, se va a explicar el proceso en el ejemplo de la cinemática montada para el *checkpoint* 5.

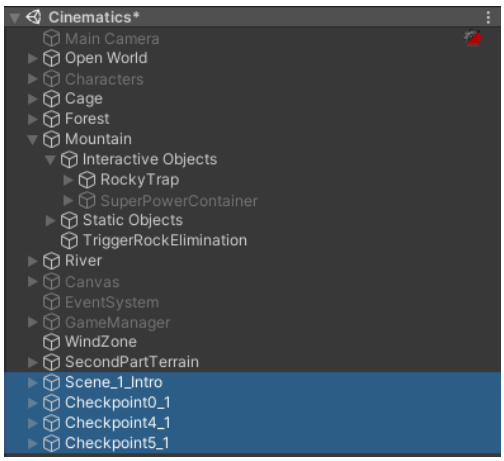

Figura 21. Objetos que contienen una línea de tiempo para cada cinemática.

Para esta cinemática se necesita el personaje de Rocky, el personaje de Phiby y un `Canvas` para situar los subtítulos.

Se pueden utilizar diferentes tipos de pistas o *tracks* en la línea del tiempo. Para añadir una pista basta con hacer clic derecho en cualquier punto de la ventana de `Timeline`. Los tipos de pistas que se han usado en este proyecto son los siguientes:

- `Animation Track`: añade una pista de animación. Es necesario indicar qué objeto es el que se va a animar. Si el objeto seleccionado no tiene un componente `Animator`, se creará uno.
- `Audio Track`: se debe seleccionar el origen del audio. Las pistas que se añadan a ese *track* se reproducirán desde ese origen.
- `Subtitle Track`: Usado para los subtítulos. Se debe seleccionar el objeto de texto deseado. Los subtítulos usados en este proyecto se han creado como explicado en el foro de Unity [34].
- `Recorder Track`: se puede añadir para que se grabe durante el tiempo que se desee.

Por último, los ajustes en las pistas de animación se realizan de la misma forma que si se tratase de una pista de edición de video o audio como muestra la Figura 22 (en este caso se han ajustado posiciones de personajes, animaciones de objetos y la aparición de subtítulos). Se dispone de un botón de grabar en cada pista. Mientras se está grabando, todos los cambios que se realicen sobre el objeto de esa pista serán grabados en los tiempos en los que se encuentre el marcador.

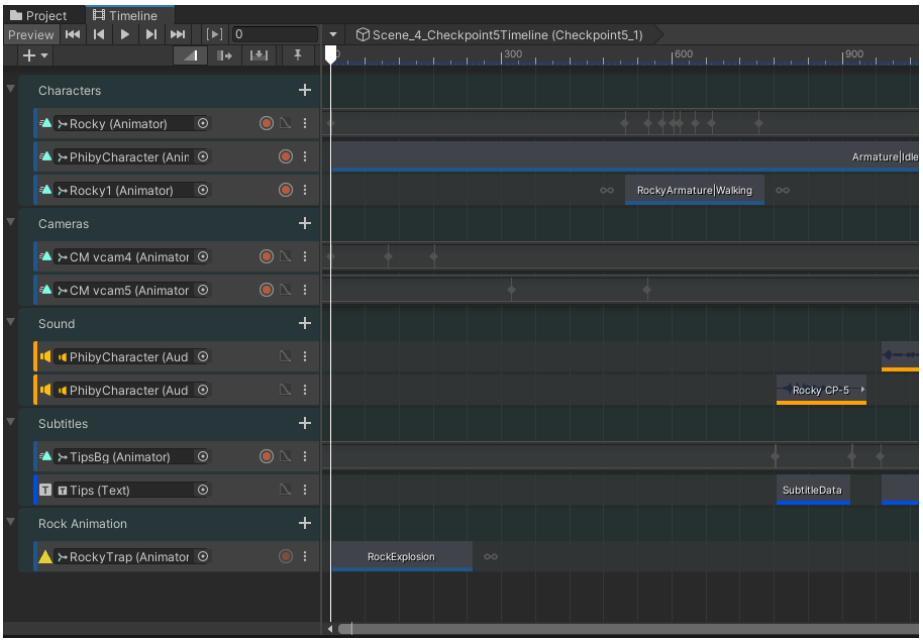

Figura 22. Línea de tiempo de `Timeline` una vez ajustados todos los elementos de la cinemática.

Merece la pena destacar la funcionalidad de hacer grupos de pistas (*Track Group*), facilitando el trabajo y manteniendo el entorno de desarrollo ordenado. En la Figura 22 se muestran grupos creados en esta parte para diferenciar entre animación de cámaras, animación de personajes, animación de subtítulos, reproducción de sonido y otras animaciones.

5.2.2. Configuración de las cámaras: uso del paquete Cinemachine

Para las cámaras (el único componente que no se ha mencionado hasta ahora) se va a hacer uso del paquete `Cinemachine` [35]. `Cinemachine` es un conjunto de módulos para operar la cámara de Unity. Soluciona así la compleja lógica detrás de, por ejemplo, el seguimiento de objetivos o la composición y transición entre escenas. De esta forma se reduce el número de ajustes manuales de la cámara (como determinar su posición) y la elaboración de scripts que tanto tiempo consumen durante el desarrollo de videojuegos.

Además, `Cinemachine` ajusta automáticamente su comportamiento para obtener el mejor plano en cada momento. Es decir, si se ajusta una animación, un terreno, etc., el usuario no deberá realizar ninguna acción adicional para ajustar la cámara.

Este paquete es útil para una gran variedad de juegos como pueden ser FPS (*First Person Shooter*), en tercera persona, o 2D. Esto se debe a que puede soportar la cantidad de cámaras que sean necesarias en la escena con un bajo coste computacional, y su comportamiento se combina a la perfección con otros módulos de Unity como pueden ser `Timeline`, animación y *assets* de postproducción. Para obtener este módulo simplemente es necesario instalarlo desde el administrador de paquetes en el software de desarrollo de Unity.

El funcionamiento de `Cinemachine` es simple: dirige una cámara de Unity para obtener varios planos. Estos planos se consiguen posicionando cámaras virtuales, las cuales mueven la cámara de Unity y controlan sus parámetros. Estas cámaras virtuales consumen pocos recursos, así que lo óptimo es usar una cámara virtual para cada plano. Para moverlas o rotarlas se debe seguir el mismo procedimiento que con cualquier otro objeto. Por último, hay que mencionar que las cámaras virtuales tienen la posibilidad de seguir o fijarse en un objetivo, dividida en dos opciones:

- `Follow`: se debe especificar un objeto con el que la cámara virtual se moverá a la par.
- `Look At`: se debe especificar un objeto al que la cámara virtual apuntará.

Es recomendable leer la documentación de esta clase para más información sobre las utilidades de `Cinemachine`, así como consejos de composición de planos [36].

El primer paso entonces para grabar una cinemática es crear un objeto de cámara de Unity. Una vez creada, se le deberá adjuntar el componente `CinemachineBrain`, mostrado en la Figura 23: este componente controla las cámaras virtuales activas en la escena y escoge la cámara virtual que se visualizará desde la cámara de Unity (`Live Camera`). También permite realizar algunos ajustes adicionales como por ejemplo la customización de efectos de mezcla entre cámaras virtuales (`Default Blend` y `Custom Blends`), personalizable para todas las cámaras virtuales existentes en la escena. Sin embargo, en este proyecto no se ha usado ningún ajuste diferente de los mostrados en la figura, pues no ha sido necesario para conseguir el resultado requerido.

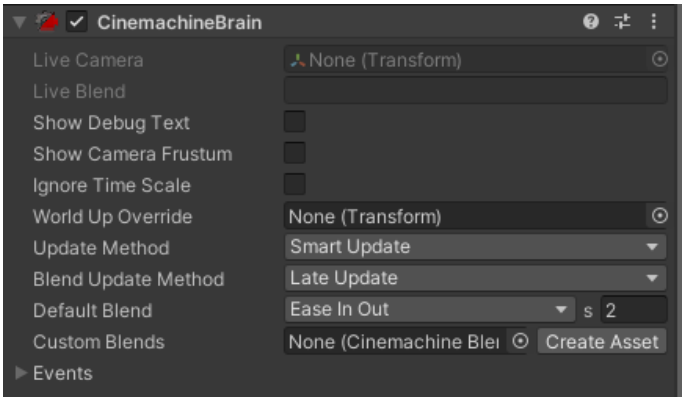

Figura 23. Componente `CinemachineBrain`.

La manera más fácil de controlar qué cámara virtual se usa en cada momento es añadiendo una pista de `Cinemachine` en la línea de tiempo de Timeline creada en el apartado anterior (y adjuntando la cámara de Unity con el componente de `CinemachineBrain`). La Figura 24 muestra el aspecto de una pista de `Cinemachine`, la cual permite decidir qué cámara virtual se usa en cada momento.

Si se quiere animar el movimiento de alguna cámara virtual, basta con animar su posición en una pista de animación; también permite hacer fundidos y mezclas entre cámaras virtuales. En la Figura 25 se puede observar el posicionamiento de las cámaras virtuales en la escena (aquellas de color rojo), de forma que cada una contiene un plano que se verá desde la cámara principal cuando se indique en la línea de tiempo de la Figura 23.

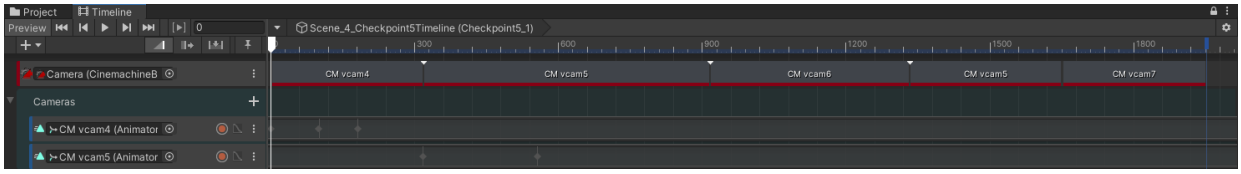

Figura 24. Colocación de cámaras virtuales de la cinemática del *checkpoint* 5 en una pista de `Cinemacine`.

Figura 25. Ejemplo de posicionamiento de cámaras virtuales en una escena para la grabación de la cinemática usada en el *checkpoint* 5.

Una vez terminado el montaje de las cámaras virtuales en la pista de Timeline y de haber obtenido el resultado deseado, la cinemática está lista para ser grabada.

5.2.3. Grabación de la cinemática: uso del paquete Unity Recorder

Para grabar las cinemáticas se usa el paquete de Unity `Recorder` [37], el cual se basa en la captura y guardado de datos durante la ejecución del modo de juego de esa escena.

La grabación de contenido se puede hacer de dos formas distintas:

- Desde la ventana de `Recorder`.
- Insertando una pista de `Recorder` en un objeto con una línea de tiempo o `Timeline`.

Los tipos de grabaciones que se pueden obtener por defecto (a través de los ajustes desde la ventana o el clip de `Recorder`) son los siguientes:

- Grabación de un clip de animación en formato de Unity (`.anim`).
- Grabación de video (con posibilidad de incluir el audio del juego).
- Generación de una secuencia de imágenes en diferentes formatos (`.jpeg`, `.png`, `.exr`).
- Generación de GIFs.
- Grabadora de audio en formato `.wav`.

Se pueden obtener otro tipo de grabaciones si se instalan otros paquetes específicos.

Sin embargo, solo se va a explicar la grabación de video pues es la usada para obtener las cinemáticas de este juego. Para una cinemática es necesario grabar el video y el audio de la escena. Si se grabasen por separado, sería necesario recurrir a un software externo para crear un archivo de video con ambos. La grabación de video de `Recorder` permite grabar incluyendo el audio del juego, permitiendo hacerlo de una vez y tener la cinemática lista antes.

La forma más sencilla y cómoda de grabar la cinemática es a través de una pista de tipo `Recorder` en la línea del tiempo `Timeline` de la cinemática a grabar, y ha sido la opción usada en este proyecto. Esta pista se inserta de la misma forma que el resto. Una vez insertada, se debe añadir un clip de grabación (o el número de clips deseados) haciendo clic derecho sobre la pista. En el caso de este proyecto se ha grabado cada cinemática usando solo un clip de `Recorder`. A continuación, se debe ajustar el tiempo durante el que se quiere grabar ese clip, y por último configurar los ajustes de la grabación de dicho clip en el menú `Inspector`.

Los ajustes usados para la grabación de las cinemáticas de este proyecto se visualizan en la Figura 26. Se debe seleccionar `Movie` como grabadora: esto permitirá que lo que se grabe sea un video. Para los ajustes de captura de video se recomienda que se grabe desde `Game View` para poder obtener una grabación de lo que se reproduce en la escena. La resolución y sobre todo la relación de aspecto deberán ser iguales para todas las cinemáticas (el juego se reproduce con una relación de aspecto también de 16:9). Por supuesto, es muy importante que la casilla `Include Audio` esté seleccionada para poder capturar el audio también. Por último, seleccionar la carpeta

y el nombre del archivo para guardar la grabación: `Recorder` permite personalizar el nombre del archivo de grabación, con la posibilidad de realizar tantas tomas de la misma grabación como sean necesarias (grabando de forma manual o con un clip). Todas las grabaciones de este proyecto se han guardado en la ruta *Assets* > *Resources* > *Video*.

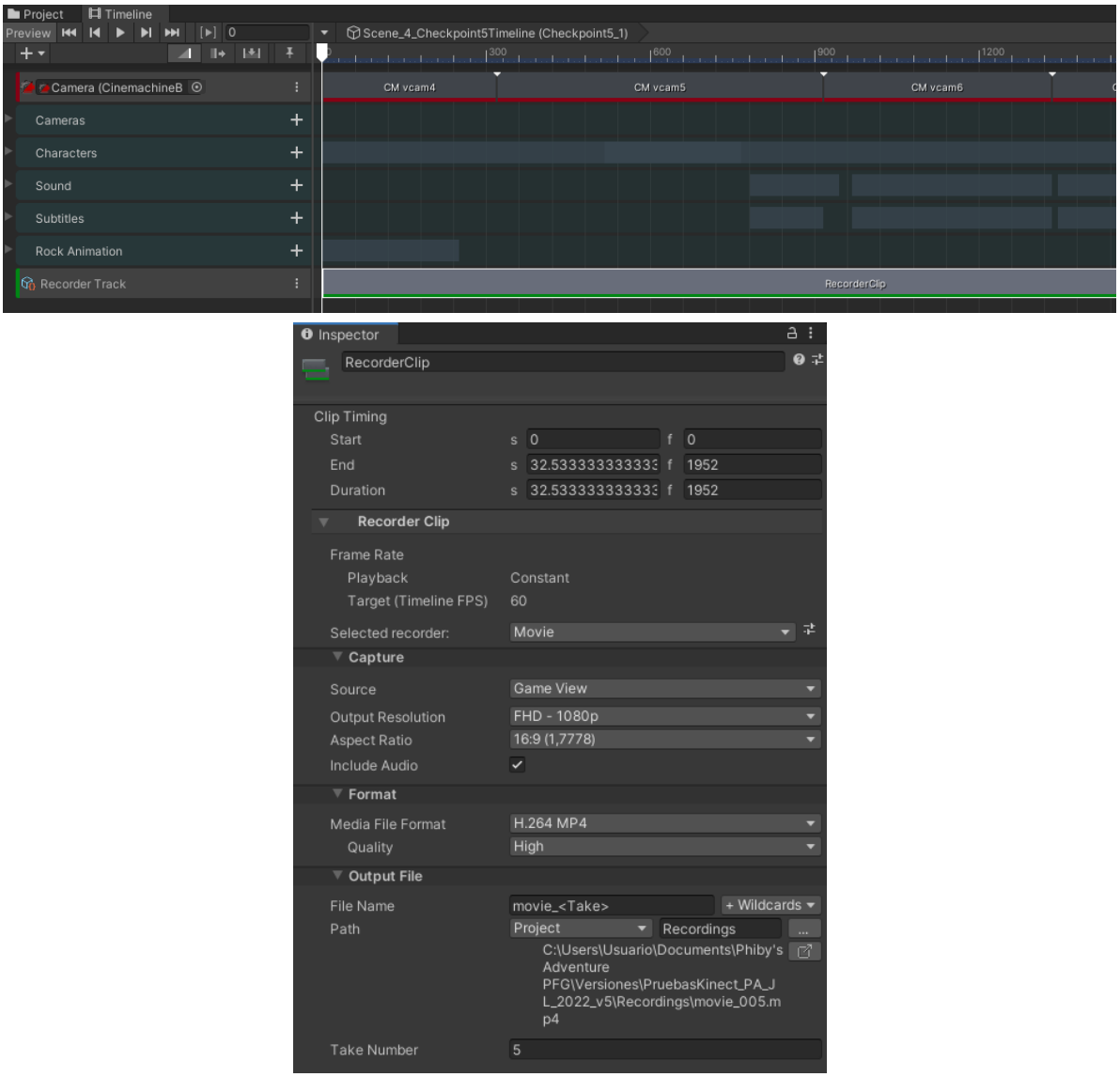

Figura 26. Arriba, clip de grabación en `Timeline`; abajo, configuración de ajustes de grabación.

A pesar de no haber sido la forma elegida, la grabación mediante la ventana de `Recorder`, mostrada en la Figura 27, también puede ser usada si se quiere accediendo desde *Window* > *General* > *Recorder* > *Recorder Window*. Los parámetros de configuración son los mismos, pero el único inconveniente es que la grabación debe ser activada y detenida manualmente, resultando menos precisa y más tediosa ya que se debe estar atento a cuándo se acaba la parte que se quiere grabar.

Una vez obtenida la grabación final de la cinemática hay que incluir ese video en la lógica del juego para que se reproduzca correctamente. Esto se explica en el apartado 5.3.

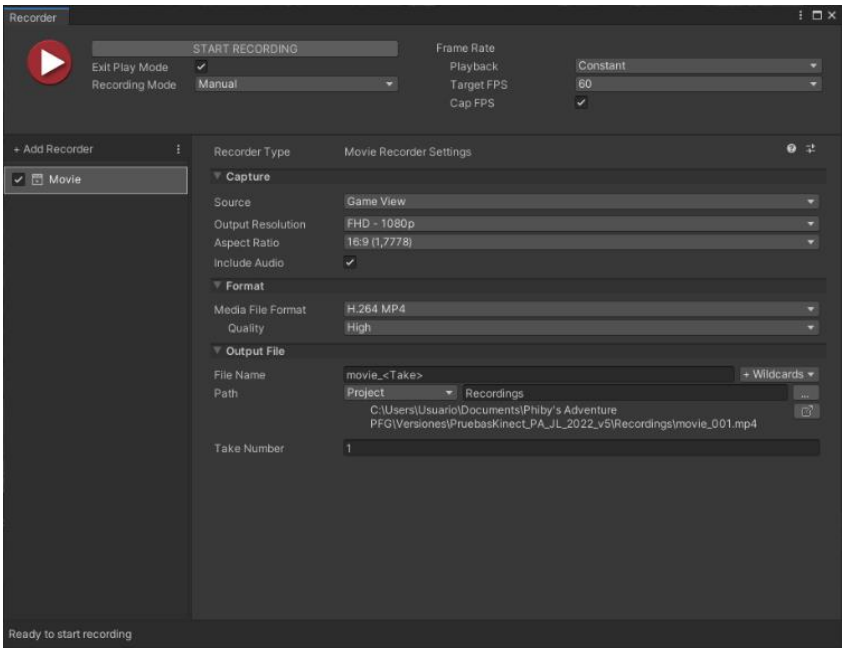

Figura 27. Ventana de grabación manual de `Recorder`.

5.3. Creación de un sistema de reproducción de cinemáticas

Para poder reproducir un video en Unity es necesario disponer de los siguientes elementos en la escena donde se quiera visualizar:

- Un objeto con un componente `VideoPlayer` (una clase propia de Unity).
- Un objeto con un componente `RawImage` (una clase propia de Unity) el cual tiene que estar incluido dentro de un objeto `Canvas`.

Dentro del objeto `Canvas` se puede incluir opcionalmente un objeto botón *skip* para poder saltar la cinemática. En este proyecto se utiliza en todas las cinemáticas.

La reproducción de la cinemática existente creada por Marta Sanz [22] estaba controlada por un script denominado `VideoPlayer.cs`. Se trata de un script sencillo que a su inicio invoca un método propio en un número determinado de segundos. Una vez transcurridos esos segundos, este método propio cargaría la escena deseada. También incorporaba el comportamiento del botón *skip* previamente mencionado (cargar la escena deseada).

Sin embargo, esta forma de reproducir un video o una cinemática dentro del videojuego sólo es eficaz para este caso. Además, el nombre del script entra en conflicto con la clase de Unity `VideoPlayer`, y se ha tenido que modificar su nombre a `VideoController.cs`. La mejora de este control de reproducción de video y cinemáticas se explica en profundidad en el apartado 5.3.2.

A continuación, se explica el sistema de reproducción de cinemáticas creado, el cual pretende ser una solución válida para la reproducción de cualquier cinemática dentro del juego.

5.3.1. Sistema de reproducción de videos de Unity

La clase y componente `VideoPlayer` [38] de Unity permite reproducir un video en la textura del objeto indicado. El video para reproducir se debe seleccionar en el parámetro `Video Clip`. Como la cinemática debe ocupar la pantalla completa, se debe usar un objeto acorde; para ello se creará en el `Canvas` un objeto con el componente `RawImage`.

Se ha mantenido el objeto de `RawImage` usado en la escena `Intro`, de cuando se reproducía la antiguan cinemática de inicio. Para reproducir en la isla se ha copiado ese objeto y el botón de *skip* y se han incluido en el `Canvas` de la escena isla. La configuración introducida mediante el Inspector de ambos componentes se muestra en la Figura 28.

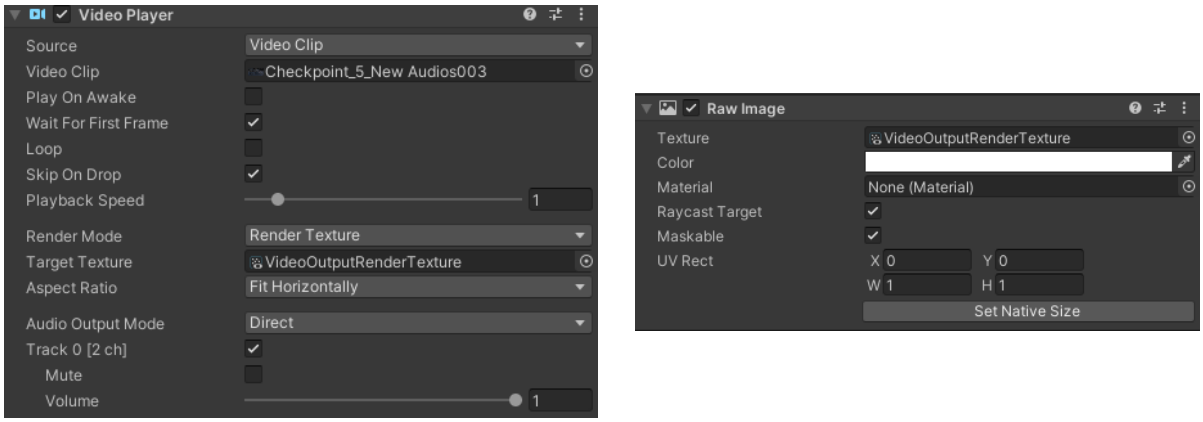

Figura 28. Configuración de los componentes `VideoPlayer` y `RawImage`.

5.3.2. Script VideoController.cs

La reproducción de las cinemáticas tiene que controlarse, ya que se trata de reproducir un video en un momento concreto del juego y, una vez que finalice, evitar que se vuelva a reproducir y que el juego transcurra su curso.

Para ello se ha desarrollado un script a partir del existente, pero más complejo: el script `VideoController.cs`. Este script se debe colocar en el mismo objeto que tenga el componente `VideoPlayer` (más adelante se detallará); además, salvo para la cinemática de inicio, ese objeto deberá tener un componente *collider* para que se pueda detectar cuándo el jugador ha entrado en esa área. Es necesario añadir los objetos que se van a necesitar en el inspector: el objeto que actúa de pantalla y el botón *skip*.

El script nada más inicializarse busca el componente de `VideoPlayer` (para poder usar sus variables desde el código). Después, activa la opción de reproducir el video nada más ejecutar la escena sólo si se trata de la escena de introducción. Esta decisión la toma en base a una variable booleana llamada `isThisIntro` la cual debe de ser verdadera sólo en ese caso; para el resto de las cinemáticas debe de ser falsa. A continuación:

- Si es la cinemática de introducción: se activan la textura donde se reproduce el video y el botón de *skip*, y se invoca al método que termina la cinemática cuando se haya acabado el video.

- Si es otra cinemática: al no reproducirse automáticamente se debe preparar el reproductor, y se busca el personaje principal para guardarlo en una variable. El script esperará hasta que el personaje entre en el *collider* del objeto mediante una función `OnTriggerEnter`. Una vez se detecte que el jugador ha entrado se impedirá que se mueva, se activan tanto el objeto con textura como el botón y se inicia la reproducción de la cinemática, invocando después al método que termina la cinemática cuando se haya acabado el video.

El script comprobará cada *frame* si el botón de *skip* es pulsado. Cuando es pulsado, ejecuta el método para terminar la reproducción de la cinemática.

El método que termina la reproducción de las cinemáticas primero detiene la reproducción, y luego actuará según si la cinemática es la de introducción o es otra:

- Para la cinemática de introducción cargará directamente la escena del menú de inicio.
- Para el resto de las cinemáticas, se desactivan tanto el objeto con la textura y el botón, y se permite al jugador volver a poder moverse. Además, se desactiva el *collider* que reproduce la cinemática para evitar que se vuelva a ver.

Para la cinemática del *checkpoint* 4 es necesario añadir en el inspector el objeto `TrigerRockElimination`. Este objeto contiene el *collider* que, cuando detecta al jugador, da paso al ejercicio de liberar a Rocky. Como se quiere que primero se reproduzca la cinemática y, a continuación, se entre al ejercicio, el *collider* del ejercicio esta inicialmente desactivado. Es por eso que al terminar la cinemática se debe volver a activar mediante este script.

Además, al terminar la cinemática del *checkpoint* 5 se activa el *collider* de Rocky que da pie a un diálogo igual que ya existía en la versión anterior. Se ha mantenido porque se considera que aporta fluidez a la transición, pero actualizando esa línea de diálogo con para incluir la voz del nuevo actor de doblaje que interpreta a Rocky, Juan Fuentes-Pila.

La inmovilización del jugador es complicada debido a que el personaje principal, Phiby, se crea cuando se ejecuta la escena. En cinemáticas como la inicial o la del *checkpoint* 5, donde se reproduce nada más cargar la escena, la creación del personaje no es lo suficientemente rápida como para que el script de `VideoController.cs` lo encuentre. Para estos casos se desactiva el movimiento de Phiby desde el script que controla el movimiento de Phiby (`PhibyMixedController.cs`): cuando se inicia este script si el jugador está en uno de esos dos *checkpoints* el movimiento se deshabilita, y, una vez acabada la cinemática, se vuelve a habilitar.

5.3.3. Añadir una cinemática a una escena

Una vez obtenida la grabación y entendido el script de control, es momento de añadirlo a una escena. Para eso se debe crear, primero, un objeto vacío dentro del objeto `Cinematic` para poder tener todas las cinemáticas del juego agrupadas y accesibles.

Este objeto vacío (nombrado con un nombre reconocible) deberá tener los siguientes componentes, asignándoles los objetos que requieran, mostrados en la Figura 29:

- El script `VideoController.cs`.
- Un componente `VideoPlayer.cs`.
- Un *collider* para poder detectar al jugador. Es importante que actúe como *trigger* y que tenga una forma adecuada para que el jugador no pueda evitarlo. Habrá que mover el objeto para poder situar el *collider* donde se desee.

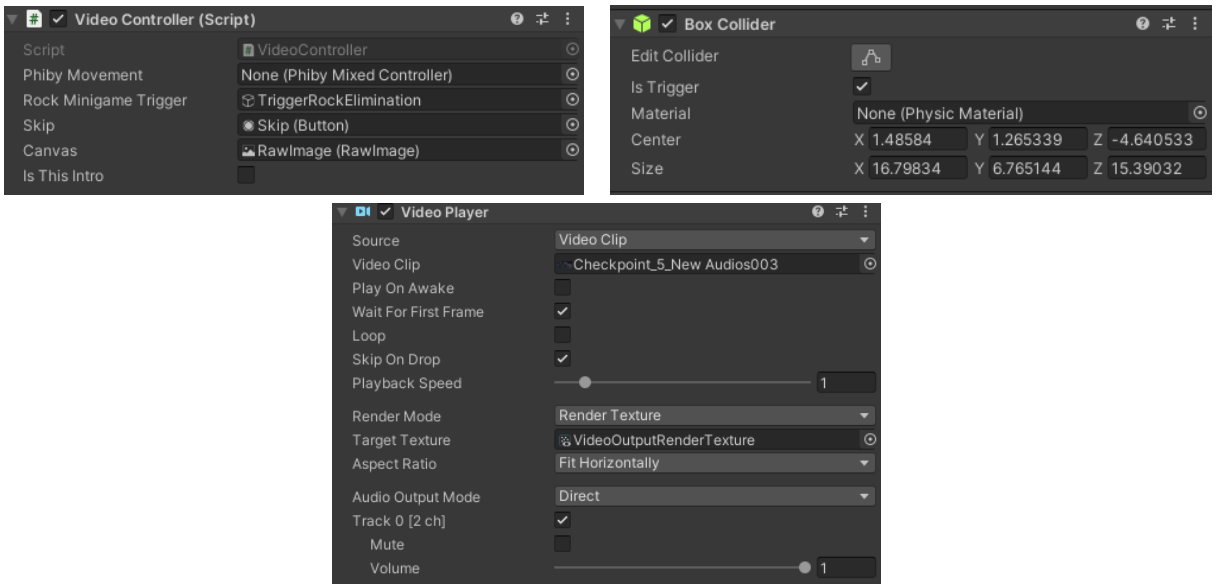

Figura 29. Configuración de componentes para introducir una cinemática en una escena.

En este proyecto es necesario tener las cinemáticas de la historia en una misma escena, en este caso, la de la isla. Como es lógico, las cinemáticas solo deberían poder ser vistas por el jugador en el momento que el desarrollador quiere.

Para automatizar este proceso y que futuros desarrolladores encuentren el proceso de añadir una cinemática más llevadero, se ha incluido en el script de `GameManager.cs` dos funciones que controlan la activación de los objetos de la cinemática:

- `CinematicReset()`: se ocupa de desactivar todos los objetos de cinemáticas. Se llama cada vez que se carga la escena de la isla.
- `CinematicActivation()`: se ocupa de activar el objeto de cinemática correspondiente al *checkpoint* en ese momento. Se llama cada vez que se carga la escena de la isla y en cada *frame* para cuando se cambia el *checkpoint* durante la misma escena.

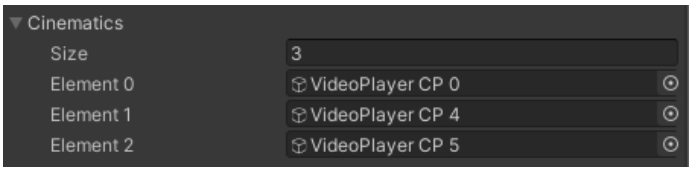

Figura 30. Configuración de los objetos de cinemática en el `GameManager`.

Para que estas funciones se ejecuten automáticamente habrá que indicar en el inspector donde esté el script de `GameManager.cs` cuáles son los objetos de las cinemáticas, como indica la Figura 30.

6. Mejora de la interfaz del menú de inicio

El menú de inicio de un juego es la carta de presentación de este, y precisamente por eso es muy importante la primera impresión que pueda causar en el jugador. La calidad interfaz de usuario que presente este menú es importante, ya que de ello depende que la navegación por los diferentes menús sea accesible y fluida, así como el grado de interés que despierta en el jugador por el juego. Si la cantidad de opciones resulta abrumadora o el menú no es claro o es complicado el jugador puede perder el interés por el juego.

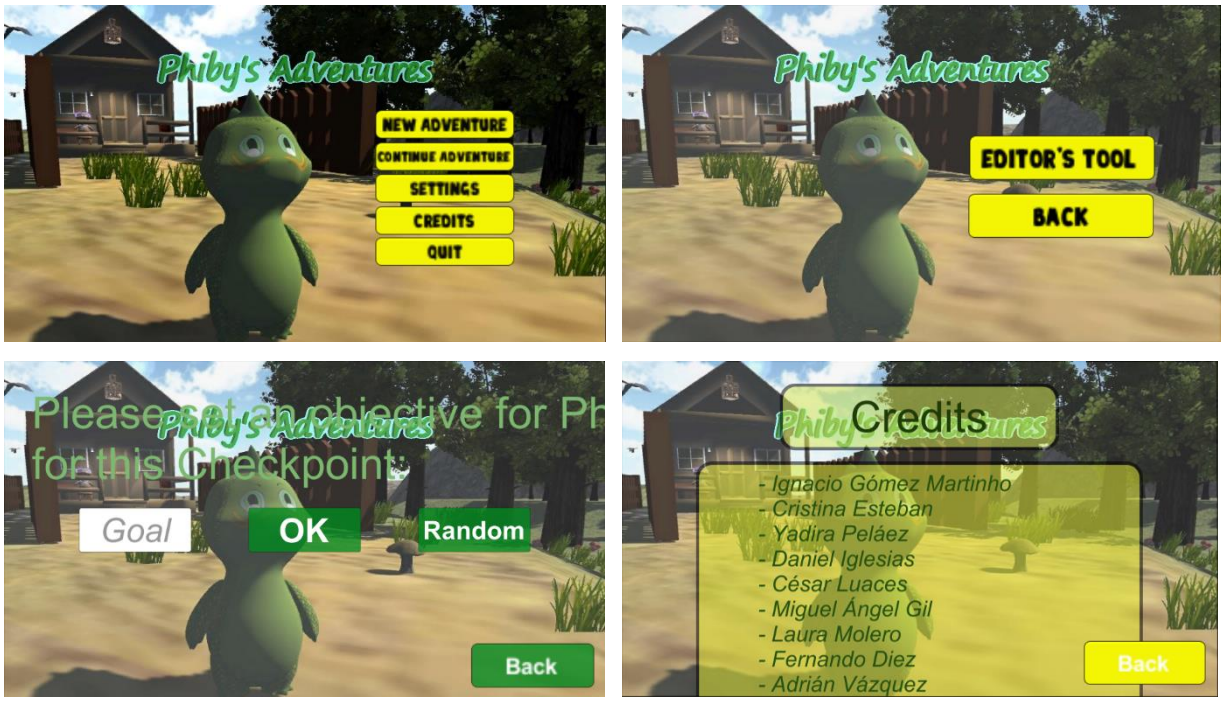

Figura 31. Pantallas del menú antiguo.

El menú principal del juego presentaba varios problemas que hacían su navegación innecesariamente complicada y con un flujo de navegación un poco frustrante, mostrados en la Figura 31:

- Títulos de los botones pixelados.
- Textos fuera de pantalla, haciéndolos imposibles de leer.
- Botones de tamaño inadecuado y poco intuitivos.
- Una fuente de texto inadecuada para ser leído a distancia (se debe tener en cuenta que el jugador se encuentra al menos a un metro de distancia a la pantalla).
- Utilización de distintas fuentes, lo cual hace inconsistente el estilo del menú.

Los dos primeros problemas fueron causados por no editar y colocar los elementos del `Canvas` en modo 2D. Es importante que siempre que se modifiquen elementos que sólo se van a ver en 2D (como un menú o el HUD (*Heads-Up Display*)) se haga en el modo 2D de Unity. Si no, los elementos no quedan bien centrados y ocurren errores de posicionamiento.

El tercer error se arregla pensando en una mejor distribución de los botones. Para empezar, se mueve el botón de salida del videojuego a la esquina izquierda y se colorea de rojo. De esta forma ese botón se distingue de los demás (es un botón con una función muy importante) y el color ayuda a diferenciarlo de los demás de forma más rápida. Además, se ha optado por cambiar la tipografía del menú a una más discreta y legible [39]. Estos cambios se muestran en las imágenes de la Figura 32.

La implementación de la lógica necesaria para el funcionamiento del menú ha sido revisada e implementada por Juan Fuentes-Pila [1], así como el diseño de la lógica que permite acceder a las escenas de ejercicios directamente desde el menú y la función de calibración de la Kinect.

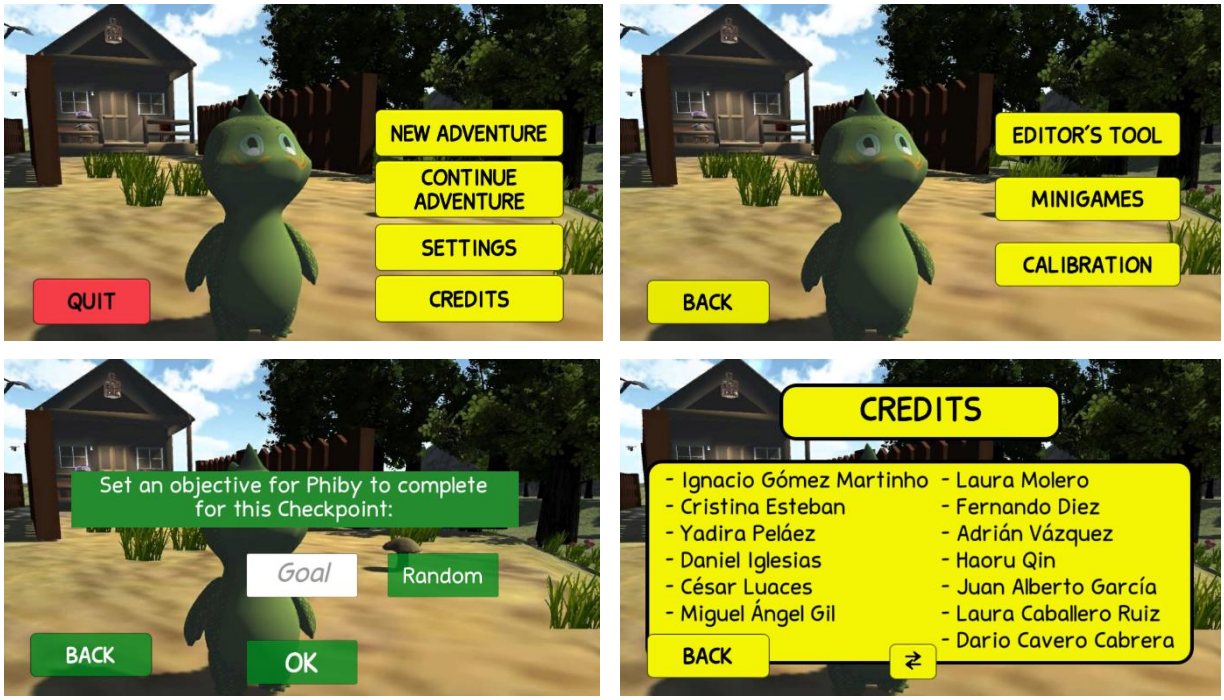

Figura 32. Pantallas del menú mejorado gráficamente.

Además, se ha añadido una nueva opción de menú que permite acceder a los minijuegos de forma individual sin depender de la historia principal, mostrada en la Figura 33. De esta forma los terapeutas pueden disponer de los ejercicios de una forma funcional cuando lo requieran.

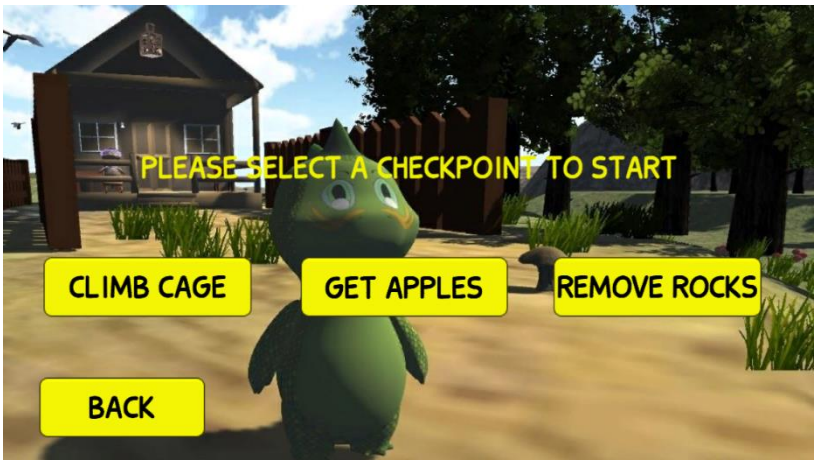

Figura 33. Pantalla de selección de carga de ejercicio.

7. Audios de realimentación

A lo largo de la realización de las diversas pruebas que han sido necesarias durante el transcurso de este proyecto, se ha observado que el jugador no dispone de ningún tipo de realimentación sonora al recoger un objeto.

En la versión inicial de “Phiby’s Adventures 3D” con la que se inició este proyecto, cuando el jugador recogía un objeto éste desaparecía y, para indicar que el jugador lo poseía, se mostraba un icono al lado del mini mapa, como muestra la Figura 34. Sin embargo, Juan Fuentes-Pila [1] ha creado, paralelamente al desarrollo de este proyecto, una nueva ventana que muestra un sistema de misiones y el mapa de la isla (para acceder a esta función se debe levantar una mano en modo Kinect o pulsar la letra M en el teclado). Con motivo de esta actualización de la interfaz de usuario se decidió incluir el inventario en esta nueva ventana, como muestra la Figura 35, y ya no se muestra el icono al lado del mini mapa como antes, dejando al jugador sin ninguna realimentación para acordarse o saber que dispone de ese objeto en el inventario.

Figura 34. Iconos de objetos recogidos por el jugador en la versión anterior del juego.

Figura 35. Iconos de objetos recogidos por el jugador en la versión actual del juego.

Se decide añadir un sonido [40] [41] a los coleccionables que recoge el jugador hasta ahora durante el juego: los fardos de paja de la jaula, las setas que hay por toda la isla, la llave de la casa de la abuela, la cuerda y el hacha. El funcionamiento que tienen los objetos en el juego es el siguiente: disponen de un *collider* (de malla o de caja, no importa de qué tipo, pero se recomienda que sea de caja para que el jugador no tenga que estar casi encima del objeto) y un script. Este script se encarga de detectar cuándo el jugador entra en el *collider* de ese objeto y, cuando lo hace, destruye el objeto o lo desactiva.

Idealmente en el futuro todos los sonidos de objetos deberán estar controlados por un mismo script en el modelo de Phiby y usando su componente `AudioSource`. Debido al tiempo limitado del proyecto debido a todos los temas abarcados en él, se ha implementado una solución sencilla que permite que el jugador tenga realimentación y sea fácil en el futuro cambiarlo: uso de un componente `AudioSource` en los objetos que reproduzca el sonido cuando el jugador entre en el *collider* de ese objeto.

Como se muestra en la primera imagen de la Figura 366, para los fardos de paja se creó un nuevo objeto vacío llamado `audioSource`, colocado en la misma posición que los fardos, con un componente de *AudioSource*. Se modifica el script `Straws.cs` (segunda imagen de la Figura 36) para poder usar el componente *AudioSource* de `audioSource` y que se reproduzca el sonido de recogida del objeto a través de él: de esta forma, aunque se desactiven los fardos, el sonido sí se reproducirá.

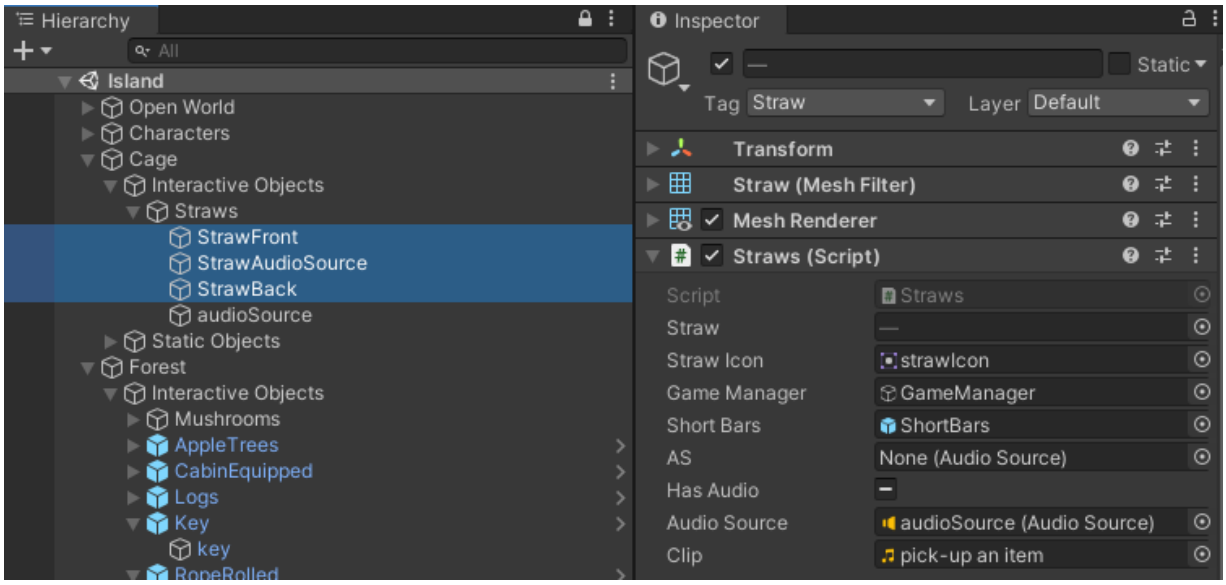

Figura 36. Configuración para incluir sonido de realimentación de los objetos `AudioSource` y los fardos de paja.

El método a seguir es el mismo para añadir el sonido a las setas, según muestra la Figura 37. Sin embargo, hay muchas setas y el hacerlo a mano resulta muy tedioso, por eso la necesidad de unificar el control de estos audios. El script modificado ha sido `GroundMushroom.cs`.

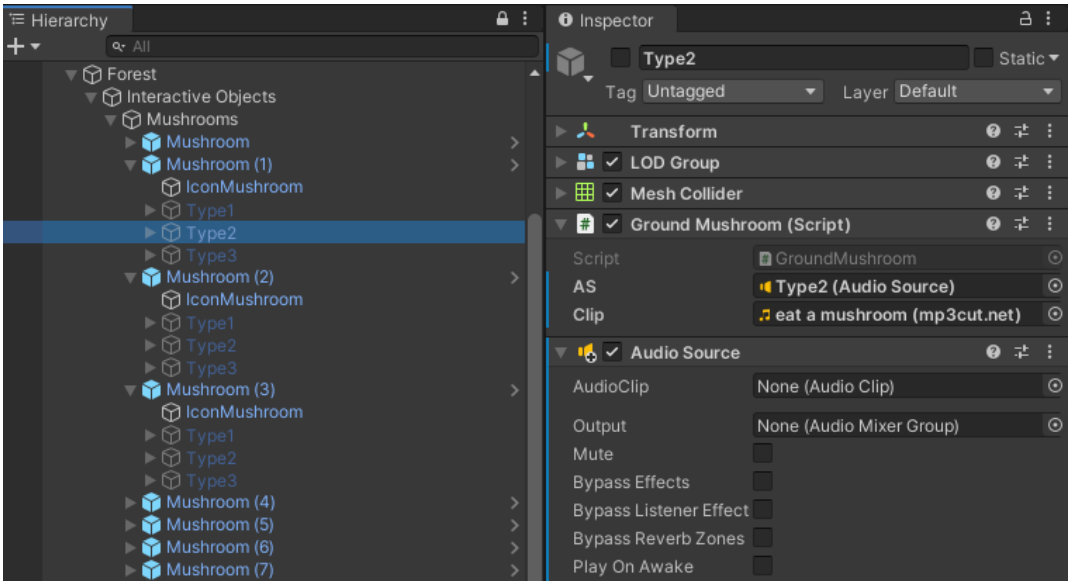

Figura 37. Configuración para incluir sonido de realimentación en las setas de la isla.

La llave, la cuerda y el hacha son *prefabs*, y ha sido necesario modificar los dos últimos. Para la llave se ha aprovechado el `AudioSource` del objeto padre de su *prefab*, añadiendo la reproducción del audio en su script `KeyController.cs`, según muestra la Figura 38. Para la cuerda (Figura 39) se ha aprovechado los componentes que había antes en su *prefab* y se ha

modificado su script `RopeController.cs` de forma que en vez de destruir el objeto hijo (el modelo de la cuerda) se desactiva. Por último, el *prefab* del hacha sólo se compone de un objeto con sus componentes, como muestra la Figura 40. Se modifica su script `AxeController.cs` para que desactive el componente de *collider* que usa y su malla (`MeshRenderer`) y se añade un componente `AudioSource` para reproducir el sonido.

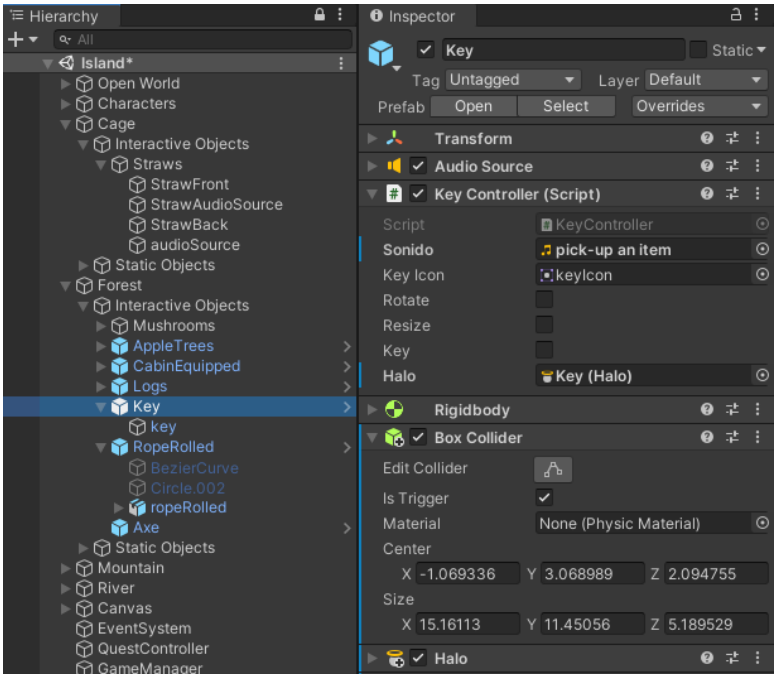

Figura 38. Configuración para incluir sonido de realimentación en la llave.

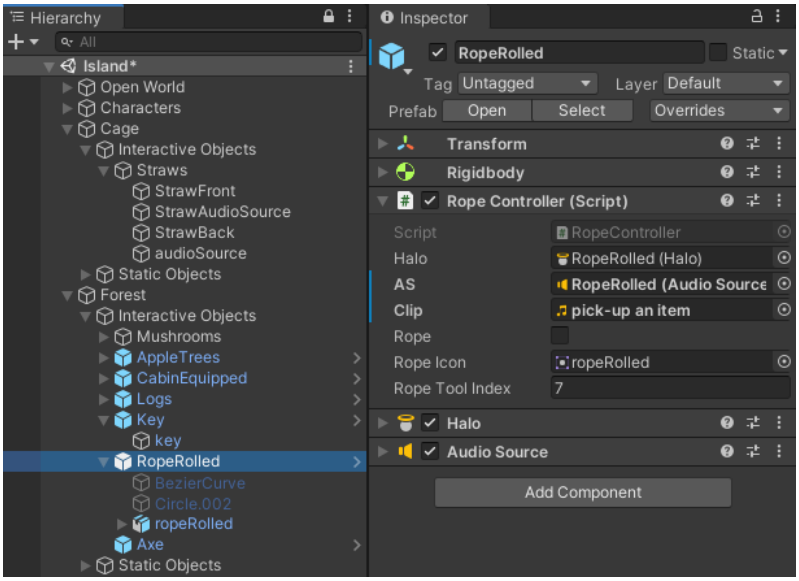

Figura 39. Configuración para incluir sonido de realimentación en la cuerda.

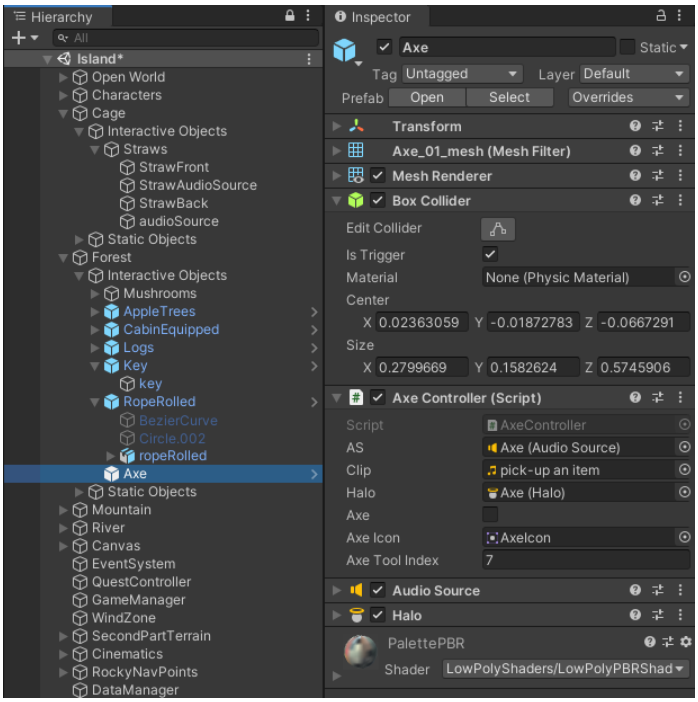

Figura 40. Configuración para incluir sonido de realimentación en el hacha.

8. Actualización de scripts

En este capítulo se describen los problemas encontrados respecto al estado de la lógica general del proyecto y las mejoras llevadas a cabo.

8.1. Limpieza de scripts obsoletos o con variables inactivas

Durante la fase de familiarización con el proyecto se revisó minuciosamente el proyecto para entender bien sus funcionalidades y el flujo de los datos y los scripts de programación. Es entonces cuando se detectan varios problemas:

- Scripts con variables que no son usadas
- Scripts que no aportan nada en ningún momento del juego
- Scripts obsoletos que no se usan

Se ha procedido a eliminar aquellos scripts que se han considerado innecesarios y a la limpieza de variables inútiles para agilizar el juego.

Un detalle para tener en cuenta es que, en los scripts, a la hora de buscar objetos o componentes de la escena hasta ahora se había estado usando el método `Find(string)` [42] (siendo el parámetro `string` el nombre del objeto o componente a buscar): consiste en buscar el objeto o componente cuyo nombre coincida con el `string` de entre todos los existentes en la escena. Este método de búsqueda consume muchos recursos al tener que buscar y comparar objeto por objeto.

Es por eso por lo que, cuando sea posible (ya que en el caso de objetos que, por ejemplo, se instancien o se creen una vez iniciada la escena no lo sería) se deben buscar los objetos mediante el inspector de Unity. De esta forma se consumirán menos recursos.

Para poder asignar variables al inspector, pero seguir manteniéndolas en privado en los scripts, será necesario usar la siguiente denominación antes de la clase del objeto: `[SerializeField] private`.

8.2. Mejora de la comunicación de la aplicación con la web

Para el desarrollo de la escena de ejercicio para liberar a Rocky fue necesario estudiar a fondo el funcionamiento de la comunicación entre la web de Blexer-med y el juego. En este apartado se va a explicar con algo más de profundidad el funcionamiento de la comunicación entre el motor de juego Unity, el *middleware* K2UM, la Kinect y la web de Blexer-med y qué cambios se han realizado para optimizar el flujo de la información.

Una vez iniciado el juego cuando ya se ha activado el *middleware* se establece una conexión UDP para enviar una serie de mensajes: mensaje de que las articulaciones están activas, un mensaje con el nombre de usuario y un mensaje de solicitud de datos de movimiento (enviado cada vez que se recibe una nueva posición de las articulaciones). Esto es posible gracias al *asset* Kinect Reciever usado en este proyecto, que también hace posible la interpretación de los datos recibidos desde el *middleware* [43].

El *middleware* sirve como conexión entre el servidor de Blexer-med y el ordenador del paciente, y como puente entre el juego y la Kinect v2. Siempre se debe ejecutar mientras está el juego en funcionamiento: descarga los datos necesarios para la realización de los ejercicios y sube los resultados del paciente; también actualiza los datos de las posiciones de las articulaciones.

El flujo de trabajo consta de cuatro momentos clave:

1. Al iniciar el *middleware* pide y descarga los datos del usuario al servidor (los parámetros asignados por el terapeuta) si previamente ha iniciado sesión al iniciar el *software*. La ruta donde se guardan los parámetros descargados es en *Assets/Resources/Parameters* con formato `.json`.
2. Cuando se inicie el juego, el *middleware* carga los parámetros guardados en el paso anterior en los datos del juego (`WebData.cs`). Si no hubiese iniciado sesión se obtienen unos parámetros por defecto descritos en el archivo `DefaultParameters.json`, situados en la misma ruta anterior.
3. Según se van completando los ejercicios, se van guardando los resultados obtenidos por el paciente en un archivo con formato `.json`, identificado por el nombre y apellidos del paciente si ha iniciado sesión, o si no el archivo con nombre `results.json`.
4. Una vez terminada la ejecución del *middleware*, éste comprueba si existen datos guardados. En el caso de que existan, se suben a la web para que el terapeuta pueda revisarlos y se borra el fichero.

En la versión anterior de este proyecto esta comunicación estaba llevada a cabo por varios scripts, siendo los más importantes los scripts `ExerciseResults.cs`, `SettingManager.cs`, `ConfigSetting.cs` y `exerciseManager.cs`.

La función del script `exerciseManager.cs` es la de guardar los parámetros del ejercicio que se está ejecutando en ese momento, cogiendo dichos parámetros del script `ConfigSetting.cs`, pero no realiza realmente ninguna acción relevante ya que obtiene siempre los parámetros del archivo `.json` por defecto. El script `ConfigSetting.cs` se encarga de proporcionar los métodos para obtener los parámetros de cada ejercicio. El script `SettingManager.cs` se encarga de solicitar la conexión UDP y recoger, si es el caso, los datos del usuario que se haya identificado; y el script `ExerciseResults.cs` se encarga de guardar los resultados del paciente una vez acabado cada ejercicio.

Esta comunicación no es óptima pues hay demasiados scripts interviniendo en el manejo de los parámetros: según se muestra en la Figura 41, en una escena de ejercicio `exerciseManager.cs` coge los parámetros correspondientes a ese ejercicio y, por ejemplo, el script `ClimbMovementController.cs` los utilizaría para llevar a cabo el ejercicio en vez de usar directamente los datos guardados en `ConfigSetting.cs`. Esto resulta complejo para el desarrollador a la hora de entender el flujo de la obtención y utilización de los parámetros en los ejercicios.

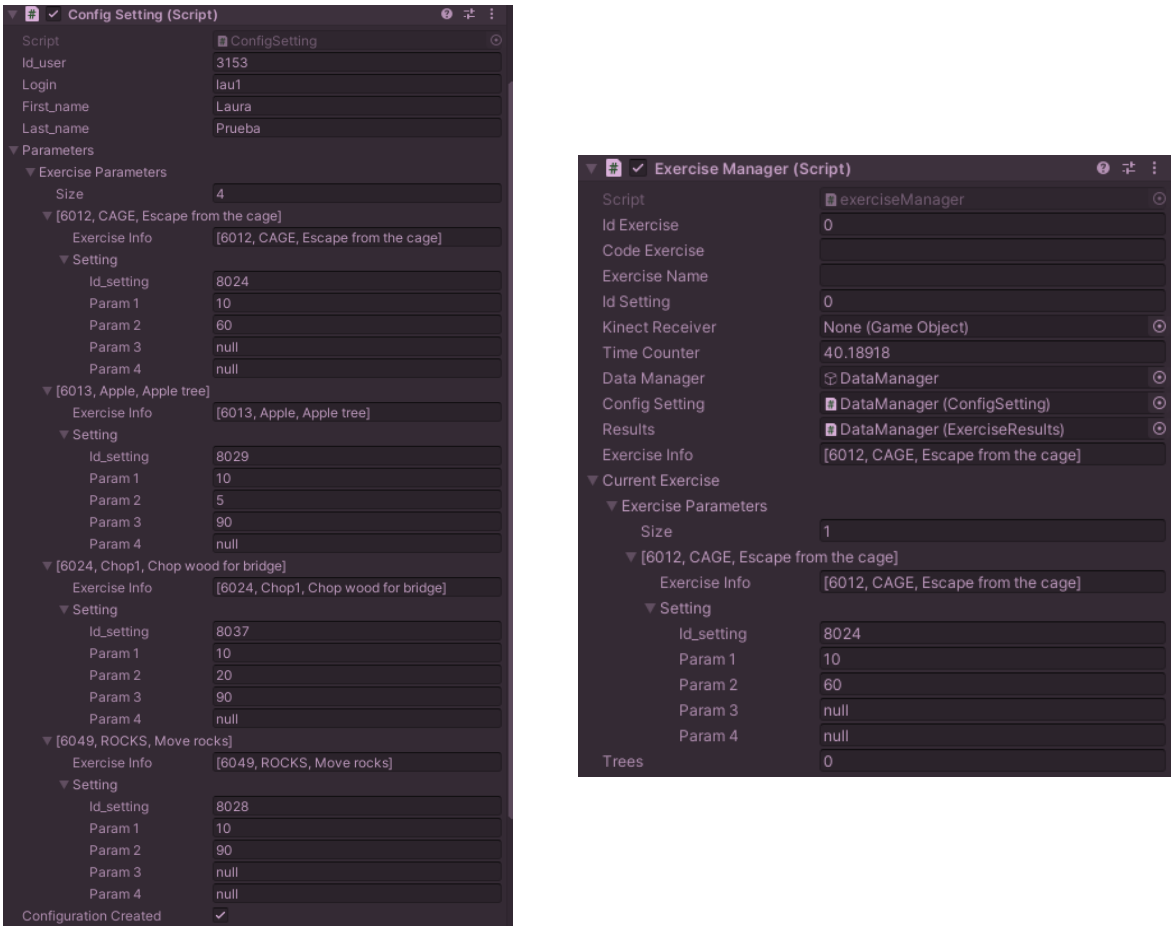

Figura 41. Variables de los scripts `ConfigSetting.cs` y `ExerciseManager.cs` en tiempo de ejecución de un ejercicio.

Es por ello por lo que se ha optado por implementar el tratamiento de los parámetros según el flujo de comunicación implementado por Eduardo Botija [44] en otro juego perteneciente al entorno Blexer-med. Consiste en la sustitución de los scripts `ConfigSetting.cs` y `exerciseManager.cs` por el uso de una estructura de datos contenida en el script `WebData.cs`, el cual permanece accesible durante toda la ejecución del juego y dispone de sus propios métodos públicos para poder acceder a los parámetros almacenados en él. Además, esta clase que almacena los parámetros va a usarse con instancias de forma que se evita la creación de variables intermedias y a la hora de programar y acceder a las variables se realizará de una forma más directa y sencilla para el desarrollador. Por último, se añade al script `WebData.cs` los métodos de guardado que implementaba el script `ExerciseResults.cs`, implementado la opción de que el nombre del archivo de guardado de los datos del usuario contenga su nombre y apellidos si previamente ha iniciado sesión a través del *middleware*.

La comunicación entre los diferentes scripts del proyecto y generación de archivos `.json` mencionados durante este capítulo se muestra en la Figura 42. La calibración de la Kinect y la actualización de los scripts de los objetos `Canvas`, `GameManager`, `CalibrationManager` y los scripts `KinectGamesManager.cs y CalibrationData.cs` ha sido implementada por Juan Fuentes-Pila [1].

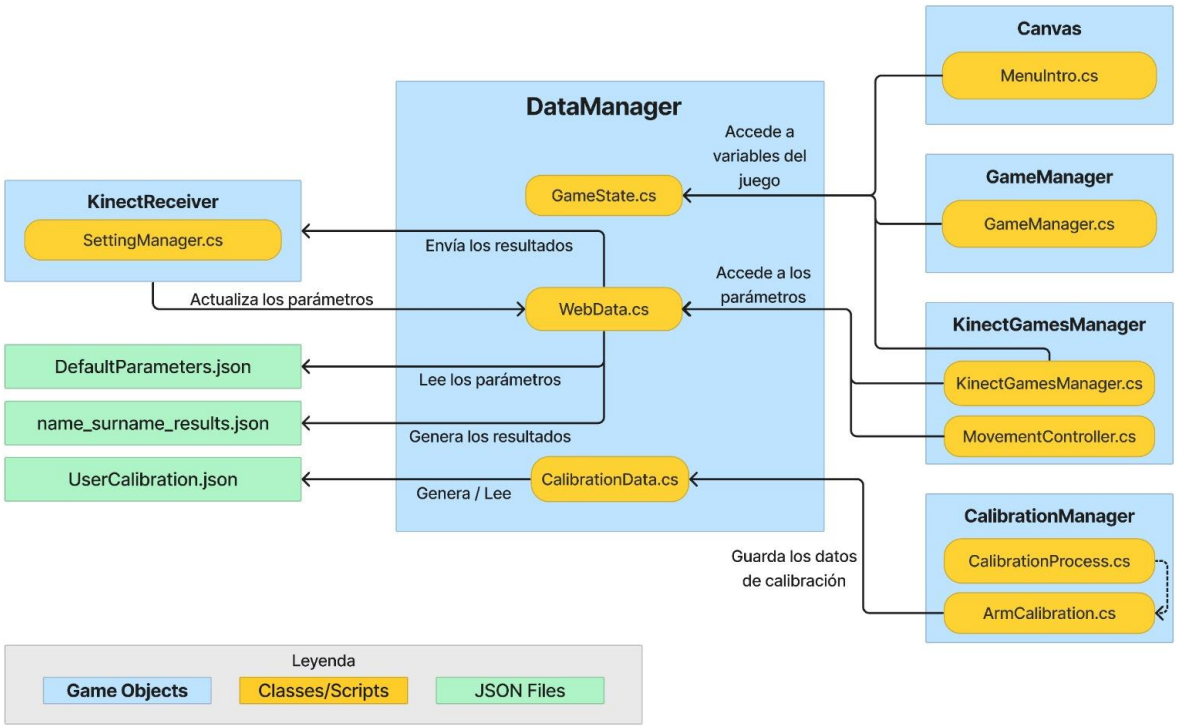

Figura 42. Diagrama de flujo de creación y acceso de los diferentes agentes (clases y objetos) [1].

9. Conclusión

Se finaliza esta memoria de proyecto de fin de carrera resumiendo brevemente el trabajo realizado, los objetivos cumplidos y los contratiempos surgidos a lo largo de su desarrollo. Se ha trabajado con una metodología iterativa e incremental debido a la naturaleza del desarrollo de videojuegos y su necesidad de varias fases de mejora, implementación, diseño y pruebas a demanda. Además, esta metodología se ha complementado con reuniones semanales junto con la tutora y directora del proyecto de “Phiby’s Adventures 3D” Martina Eckert y con Juan Fuentes-Pila, compañero que ha desarrollado también su trabajo de fin de grado en este proyecto. Estas reuniones han servido para informar de los avances realizados y tomar decisiones sobre el futuro cercano del desarrollo y la complementación de los trabajos paralelos llevados a cabo por Juan. Como resultado se han obtenido varias versiones del videojuego hasta llegar a la versión final.

Se realiza una familiarización con el proyecto de “Phiby’s Adventures 3D” y la historia de su desarrollo, leyendo la documentación de antiguas contribuciones. A continuación, se prueba el juego y junto con la tutora se establece la lista de objetivos a cumplir para este trabajo de fin de grado centrados en la mejora de la historia y de la experiencia del jugador. Uno de los mayores retos de este proyecto de fin de grado ha sido el enfrentarse a un proyecto con varios años de desarrollo y con contribuciones de muchos alumnos, pues eso resulta en un proyecto con una gran variedad de estilos de programación y metodologías de desarrollo de videojuegos.

Para mejorar una parte de la historia en la que sus mecánicas resultaban repetitivas y podían aburrir al jugador se decidió alterar la historia de forma que se pudiera incluir un nuevo ejercicio que incorporase un nuevo movimiento. A pesar de que el modelo presenta problemas a la hora de replicar los movimientos del jugador, la Kinect sí reconoce los movimientos y el ejercicio es funcional.

De cara también a mejorar la historia se decide introducir cinemáticas para hacer más dinámicas aquellas partes con más peso narrativo y evitar por ejemplo que el jugador tenga que andar distancias demasiado largas para escuchar un diálogo. Además, se mejora la cinemática de introducción ya existente dotándola de mejor calidad, así como también se desarrolla un sistema de reproducción de cinemáticas totalmente funcional y aprovechable para incluir nuevas cinemáticas en futuros trabajos.

Al influir también en la experiencia del jugador, se han escalado adecuadamente todos los objetos de la escena del menú principal, haciendo que se vean atractivos para el jugador ya que es la carta de presentación del juego. Además, se ha añadido un nuevo menú para permitir probar los ejercicios sin tener que jugar la historia entera.

Para terminar, se ha mejorado la lógica que había implementada de forma que esté más unificada y permita un mejor flujo de trabajo más fácil de comprender y manejar para los futuros desarrolladores: limpieza de scripts obsoletos, mejora del uso de métodos de programación y asignación de objetos en Unity, y la mejora de la comunicación entre el *middleware* y el juego.

Se ha de mencionar que, además de los contratiempos encontrados debido a la complejidad del código por haber recibido contribuciones de diferentes desarrolladores, se han encontrado dificultades a la hora de comprender el flujo de trabajo de los principales sistemas del juego, como puede ser el flujo de datos ente el *middleware* y el uso de esos parámetros en los

ejercicios, el manejo de los *checkpoints* y el tratamiento de variables locales o el funcionamiento de los ejercicios existentes. No ha sido hasta pasado bastante tiempo trabajando en el proyecto que se ha entendido su funcionamiento debido a que el GDD (*Game Design Document*) de este proyecto no ahonda suficiente en el funcionamiento de éste, retrasando así la consecución de los objetivos y sus correspondientes pruebas. Es por eso por lo que en este proyecto se hace especial énfasis en la explicación de los procesos y la explicación de los nuevos scripts creados y, cuando es necesario, la explicación de scripts ya existentes y relevantes para conseguir completar los objetivos de este proyecto de fin de grado, además de la actualización del GDD.

Se ha tenido que añadir más tiempo de lo planificado al proyecto por la necesidad de buscar errores debido a la programación incoherente de los estudiantes involucrados anteriores. Debido a esto, algunas ideas adicionales no se han podido realizar y se proponen para el trabajo futuro, sin embargo, se ha solucionado muchas otras cosas que aportan estabilidad al juego y está previsto que en breve se pasará una primera versión ejecutable a un colegio de integración donde lo probarán algunos niños con parálisis cerebral.

10. Futuras líneas de trabajo

Para el futuro de este proyecto se pueden explorar varias líneas de trabajo:

- Mejora de la reproducción y almacenaje de los audios del juego: lo óptimo sería implementar un script para cada personaje en el que se disponga de todos las líneas y diálogos. Desde ese script se controlaría la reproducción de los audios de cada personaje para evitar que el control del audio se maneje desde scripts cuya principal función es otra.
- Mejora del modelo de Phiby: el modelo actual se compone de un esqueleto sencillo con solamente articulaciones de hombros, cuello, columna, caderas y cola. De cara a futuras animaciones y posibles implementaciones de ejercicios, se puede beneficiar de disponer de alguna articulación más como pueden ser codos, muñecas, rodillas o tobillos.
- Implementación de ejercicios con movimientos más variados (nuevas mecánicas): hasta ahora todos los ejercicios implementados hacen usos de movimientos que consisten en levantar los brazos ya sea vertical u horizontalmente. Movimientos como realizar círculos con los brazos o con los codos pueden aportar más variedad y dinamismo a los ejercicios y a la experiencia del jugador. Ejercicios como nadar o volar pueden hacer uso de estos nuevos movimientos, pero manteniendo los ejercicios de cintura para arriba pues se pretende que el juego sea jugado por personas en sillas de ruedas.
- Mejora del cálculo de los movimientos en los ejercicios: actualmente se comprueba si se realiza un ejercicio comprobando si la mano se eleva por encima del eje de coordenadas Y. Sin embargo, puede ser más conveniente referenciar los movimientos a otras partes del cuerpo del jugador, como puede ser el hombro, o el codo. Esta mejora puede, además contribuir a implementar nuevos ejercicios con nuevos movimientos.
- Mejora de las animaciones de Rocky: se sienten poco orgánicas y se podrían mejorar.
- Ampliación de la base de datos integrada en la plataforma Blexer-med de forma que se pueda disponer de más parámetros ajustables en los ejercicios para dar más libertad a los terapeutas.
- Unificación, documentación y limpieza de scripts: aún quedan muchos scripts obsoletos y otros muchos que tienen funcionalidades similares, pero ocupan varios scripts. Esta labor mejoraría el rendimiento del proyecto y facilitaría el flujo de trabajo.
- Creación de nuevas cinemáticas en futuros momentos de desarrollo de historia: aprovechar el potencial de Unity para realizar cinemáticas que contribuyan a la inmersión del jugador en la historia. Por ejemplo, cuando el jugador esté cerca de llegar al volcán, enseñarle una cinemática en la que la cámara vuele alrededor del volcán y muestre los alrededores.
- Creación de un controlador de todas las cinemáticas.
- Creación de un método general que permita acceder a los parámetros de los ejercicios sin tener que escribir su identificación.

- Implementación de un controlador de energía. En esta versión no se cuenta con un método general que agrupe el ajuste de energía para cada ejercicio pues sólo se cuenta con el realizado en el ejercicio de recoger manzanas.
- Mejora de la descripción del juego y sus aspectos técnicos en el GDD.
- Implementar el resto de la historia pendiente.

11. Bibliografía

[1] J. Fuentes-Pila, «Mejora de la lógica e incorporación de sistemas para el videojuego terapéutico "Phiby's Adventures 3D",» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2022.

[2] Evolve, «Acerca de,» [En línea]. Available: https://evolvrehab.com/es/evolvrehab/evolvrehab_body/. [Último acceso: 2022 Junio].

[3] Mira Rehab, «Product,» [En línea]. Available: https://www.mirarehab.com/product/. [Último acceso: Junio 2022].

[4] CITSEM, «Interfaces naturales para fines terapéuticos,» [En línea]. Available: https://www.citsem.upm.es/es/investigacion/areas-estrategicas/interfaces-naturales-para-fines-terapeuticos. [Último acceso: Marzo 2022].

[5] CITSEM, «“Medical Access to the Blender Exergames, Blexer,» [En línea]. Available: https://blexer-med.citsem.upm.es/index.php. [Último acceso: Marzo 2022].

[6] M. J. Ramos, «Plataforma médica para el entorno de videojuegos terapéutico"Blexer",» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, Julio 2017.

[7] A. A. López, «Implementación de medidas de seguridad en plataforma médica para entorno de videojuegos terapéuticos,» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, Julio 2019.

[8] J. López, «Kinect v2 para Windows,» IGNEspaña, 7 Octubre 2013. [En línea]. Available: https://www.xatakawindows.com/xbox/la-evolucion-de-kinect-y-la-importancia-de-microsoft-research. [Último acceso: Junio 2022].

[9] W. Community, «Kinect for Widows,» 19 Mayo 2022. [En línea]. Available: https://docs.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows. [Último acceso: Junio 2022].

[10] C. Luaces Vela, «Diseño e implementación de un entrono virtual de ejercicios físicos,» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomuniación, UPM, Madrid, Febrero 2018.

[11] Blender, [En línea]. Available: https://www.blender.org/about/. [Último acceso: Junio 2022].

[12] M. E. C. E. Y. P. I. Gómez-Martinho, «Phiby’s Adventures v1. Memoria técnica. Registro software,» Madrid, 2018.

[13] GAMMA, CITSEM, «Phiby's Adventures 3D,» 10 Abril 2021. [En línea]. Available: https://naturalinterfaces.etsist.upm.es/lineas-de-investigacion/body-group/phibys-adventures-3d.

[14] L. M. Salazar, «Diseño e implementación del entorno para el juego serio terapéutico "Phibys Adventures v2",» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicaciones, UPM, Madrid, 2019.

[15] M. Á. Gil Gil, «Integración y calibración de movimientos captados mediante la cámara Kinect para el juego serio terapéutico "Phiby's Adventures v2",» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicaciones, UPM, Madrid, 2019.

[16] J. A. García Fernández, «Memoria de práctica externa ETS de Iingeniería y Sistemas de Telecomunicación,» Escuela Técnica Superior de Iingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2020.

[17] F. Díez Muñoz, «Memoria de Práctica Externa ETS de Ingeniería y Sistemas de Telecomunicación UPM,» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2020.

[18] J. A. García Fernández, «Diseño e implementación de una lógica de ajuste terapéutico para el juego serio "Phibys Adventures 3D",» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicaciones, UPM, Madrid, 2020.

[19] H. Qin, «Diseño e implementación de la historia, las mecánicas y el flujo del videojuego serio "Phiby's Adventures v2",» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicaciones, UPM, Madrid, 2020.

[20] L. Caballero Ruiz, «Memoria de práctica externa ETS de Ingeniería y Sistemas de Telecomunicación,» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicaciones, UPM, Madrid, 2021.

[21] D. Cavero, «Memoria de práctica externa ETS de Ingeniería y Sistemas de Telecomunicación,» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicaciones, UPM, Madrid, 2021.

[22] M. Sanz Gómez, «Mejora y progreso del videojuego terapéutico "Phiby's Adventure 3D",» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicaciones, UPM, Madrid, 2021.

[23] Unity, «Unity,» [En línea]. Available: https://unity.com/es. [Último acceso: Junio 2022].

[24] Unity, «Creación de scripts en Unity para programadores expertos,» [En línea]. Available: https://unity.com/es/how-to/programming-unity. [Último acceso: Junio 2022].

[25] Mono, «Mono-Project,» [En línea]. Available: https://www.mono-project.com. [Último acceso: 2022 Junio].

[26] Microsoft Community, «Introducción a .NET Framework,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/framework/get-started/. [Último acceso: Junio 2022].

[27] Unity, «Unity 2019.4.21,» 2019. [En línea]. Available: https://unity3d.com/es/unity/whats-new/2019.4.21. [Último acceso: Junio 2022].

[28] Visual Studio Code, «Visual Studio Code,» [En línea]. Available: https://code.visualstudio.com. [Último acceso: Junio 2022].

[29] GitHub, «GitHub,» [En línea]. Available: https://github.com. [Último acceso: 2022 Junio].

[30] Unity, «Navegación y Pathfinding (Búsqueda de Caminos),» 2 Octubre 2018. [En línea]. Available: https://docs.unity3d.com/es/2019.4/Manual/Navigation.html. [Último acceso: Junio 2022].

[31] Unity, «Construyendo un NavMesh,» [En línea]. Available: https://docs.unity3d.com/es/2019.4/Manual/nav-BuildingNavMesh.html. [Último acceso: Junio 2022].

[32] Unity, «Creando un Agente NavMesh,» [En línea]. Available: https://docs.unity3d.com/es/2019.4/Manual/nav-CreateNavMeshAgent.html. [Último acceso: Junio 2022].

[33] Unity, «About Timeline,» 10 Octubre 2018. [En línea]. Available: https://docs.unity3d.com/Packages/com.unity.timeline@1.2/manual/index.html. [Último acceso: Junio 2022].

[34] seant_unity, 9 Junio 2017. [En línea]. Available: https://forum.unity.com/threads/use-timeline-to-set-sub-title-on-ui.475564/. [Último acceso: Marzo 2022].

[35] Unity, «Cinemachine Documentation,» [En línea]. Available: https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/index.html. [Último acceso: Junio 2022].

[36] Unity Technologies, 12 Octubre 2020. [En línea]. Available: https://learn.unity.com/project/cutscenes-and-trailers-with-timeline-and-cinemachine?language=en. [Último acceso: Junio 2022].

[37] Unity, «Unity Recorder User Manual,» [En línea]. Available: https://docs.unity3d.com/Packages/com.unity.recorder@2.0/manual/index.html. [Último acceso: Junio 2022].

[38] Unity, «Video Player component,» 7 Mayo 2017. [En línea]. Available: https://docs.unity3d.com/Manual/class-VideoPlayer.html. [Último acceso: Junio 2022].

[39] Dafont, [En línea]. Available: https://www.dafont.com/es/stanberry.font. [Último acceso: Junio 2022].

[40] FreeSound, «Eating Chips.wav by ryanharding95,» 30 abril 2015. [En línea]. Available: https://freesound.org/people/ryanharding95/sounds/272440/. [Último acceso: Junio 2022].

[41] Freesound, «SFXKeyPickUp.wav by dkiller2204,» 22 marzo 2018. [En línea]. Available: https://freesound.org/people/dkiller2204/sounds/422971/. [Último acceso: Junio 2022].

[42] Unity, «GameObject.Find,» [En línea]. Available: https://docs.unity3d.com/es/530/ScriptReference/GameObject.Find.html. [Último acceso: Junio 2022].

[43] D. Iglesias Canelo, «Middleware K2UM 2.0,» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomuniación, UPM, Madrid, 2019.

[44] E. Botija, «Revisión y Mejora de Videojuego West Gun Theraphy,» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2022.

[45] CITSEM, «CITSEM,» [En línea]. Available: https://www.citsem.upm.es/es/. [Último acceso: Marzo 2022].

[46] GAMMA, «Grupo de Aplicaciones Multimedia y Acústica,» [En línea]. Available: https://www.citsem.upm.es/es/quienes-somos/grupos-de-investigacion/grupo-de-aplicaciones-multimedia-y-acustica-gamma. [Último acceso: Marzo 2022].

ANEXO I: Presupuesto

En este anexo se pretende valorar todos los recursos utilizados para la realización de este proyecto: material, personal y licencias de software. De esta forma se calculará la inversión necesaria para llevar a cabo el proyecto.

Así mismo se calculará un presupuesto mínimo para poder realizar el proyecto para que sirva de referencia a la hora de planificar futuros proyectos similares.

Presupuesto empleado

Coste personal

Se ha contado con un ingeniero de sonido e imagen junior, con 285h de trabajo estimadas al desarrollo del proyecto y 40h dedicadas a la redacción de la documentación pertinente.

Según las estimaciones de los salarios en el mercado laboral actual, el salario bruto de un ingeniero junior oscila sobre los 36.100,00€ [1]. Este salario se traduce, con una jornada laboral de 40 horas semanales en 18,80€ por hora. En total se han requerido 325h de trabajo, resultando en la suma de 6.110,00€ por el trabajo realizado.

Coste material y licencias

En el apartado material se ha necesitado un ordenador de alta capacidad computacional pues el desarrollo del proyecto así lo requiere. El ordenador de sobremesa cuenta con los periféricos de teclado, ratón y monitor, y cuenta con las siguientes especificaciones:

- Procesador AMD Ryzen 7 3800X 3.9GHz.
- RAM 3200MHz PC4-25600 32GB.
- SSD de 500GB.
- Gigabyte GeForce RTX 3050 8GB.
- Sistema operativo Windows 10.

El valor del ordenador (incluyendo fuente de alimentación, refrigeración, periféricos) asciende a un total de 1.482,83€.

Se necesita además una cámara Kinect v2 de Microsoft. Tiene un precio de 144,95€ en GAME [2]. Para conectar la Kinect a un ordenador es necesario un adaptador, valorado en 30,99€ en Amazon [3].

En cuanto al software utilizado, Unity, GitHub y Visual Studio Code tienen una opción de licencia gratuita.

Coste total

Los costes totales ascienden a un total de 7.737,78€.

Tabla 1. Presupuesto de los costes personales y materiales empleados en el proyecto.

Descripción	Unidades	Coste unitario	Horas	Coste por horas (€/h)	Total
Personal					
Ingeniero junior	1	-	325	18,80€/h	6.110,00€
				Parcial	6.110,00€
Material					
Ordenador	1	1482,83€	-	-	1482,83€
Kinect 2.0	1	144,95€	-	-	144,95€
				Parcial	1.627,78€
Licencias de software					
Unity	1	0,00€	-	-	0,00€
Visual Studio Code	1	0,00€	-	-	0,00€
GitHub	1	0,00€	-	-	0,00€
				Parcial	0,00€
Total		7.737,78€			

Presupuesto mínimo

Para abaratar costes es posible reducir las características del ordenador empleado para el desarrollo del proyecto. No obstante, es obligatorio que este ordenador utilice el sistema operativo de Windows, pues la Kinect no funciona sin él.

Unas características mínimas adecuadas para el correcto funcionamiento del software de desarrollo [4] del proyecto son las siguientes:

- Procesador Core i5 Gen12, Core i7 Gen11, AMD Ryzen 7.
- RAM 16 GB.
- SSD de 256GB.
- Tarjeta gráfica que soporte DX10, DX11, and DX12.
- Sistema operativo Windows 10.

Si se busca un ordenador portátil con estos requisitos, se puede elegir un modelo parecido al ordenador portátil Asus Zenbook 14 UM425UAZ-KI016T el cual tiene un precio de 949€ [5].

Con un ordenador menos exigente el presupuesto total sería de 7.203,95€. Este presupuesto resulta en un ahorro de 533,83€ respecto al presupuesto del material utilizado.

Tabla 2. Estimación de un presupuesto mínimo de los costes personales y materiales para el desarrollo del proyecto.

Descripción	Unidades	Coste unitario	Horas	Coste por horas (€/h)	Total
Personal					
Ingeniero junior	1	-	325	18,80€/h	6.110,00€
				Parcial	6.110,00€
Material					
Ordenador	1	949,00€	-	-	949,00€
Kinect 2.0	1	144,95€	-	-	144,95€
				Parcial	1.093,95€
Licencias de software					
Unity	1	0,00€	-	-	0,00€
Visual Studio Code	1	0,00€	-	-	0,00€
GitHub	1	0,00€	-	-	0,00€
				Parcial	0,00€
Total			7,203,95€		

Referencias

[1] Jobted, «Sueldo del Ingeniero de Telecomunicaciones en España,» [En línea]. Available: https://www.jobted.es/salario/ingeniero-telecomunicaciones. [Último acceso: Junio 2022].

[2] GAME, «KINECT XBOX ONE,» [En línea]. Available: https://www.game.es/kinect-xbox-one-xbox-one-102796. [Último acceso: Junio 2022].

[3] Amazon, [En línea]. Available: https://www.amazon.es/PeakLead-Mejorada-Adaptador-alimentación-incluida/dp/B07QK9CGTT/ref=sr_1_1?__mk_es_ES=ÅMÅŽÕÑ&crid=4E3B9Z8WSQ3Q&keywords=adaptador+usb+kinect+ordenador&qid=1655481669&s=electronics&sprefix=adaptador+usb+kinect+ordenador%2Celectronics%. [Último acceso: Junio 2022].

[4] Unity, «System requirements for Unity 2021 LTS,» 17 Junio 2022. [En línea]. Available: https://docs.unity3d.com/Manual/system-requirements.html#editor. [Último acceso: Junio 2022].

[5] PcComponentes, «Asus Zenbook 14 UM425UAZ-KI016T AMD Ryzen 7 5700U/16GB/512GB SSD/14",» [En línea]. Available: https://www.pccomponentes.com/asus-zenbook-14-um425uaz-ki016t-amd-ryzen-7-5700u-16gb-512gb-ssd-14. [Último acceso: 2022 Junio].

ANEXO II: Uso de Unity con GitHub. GITLFS

GitHub es una plataforma de desarrollo colaborativo que permite alojar proyectos y usa el sistema de control de versiones Git. A pesar de ser más conocido por ser usado principalmente para la creación y desarrollo de código fuente, en la actualidad GitHub también puede ser un buen aliado para el desarrollo de proyectos en Unity.

Para empezar a usar GitHub, lo primero que se necesita es crearse una cuenta en la plataforma como en la Figura 43. Después, según se muestra en la Figura 44, se debe crear un repositorio: es el sitio en el que se almacenará el proyecto. Se deberá escoger un nombre y, opcionalmente, incluir una descripción.

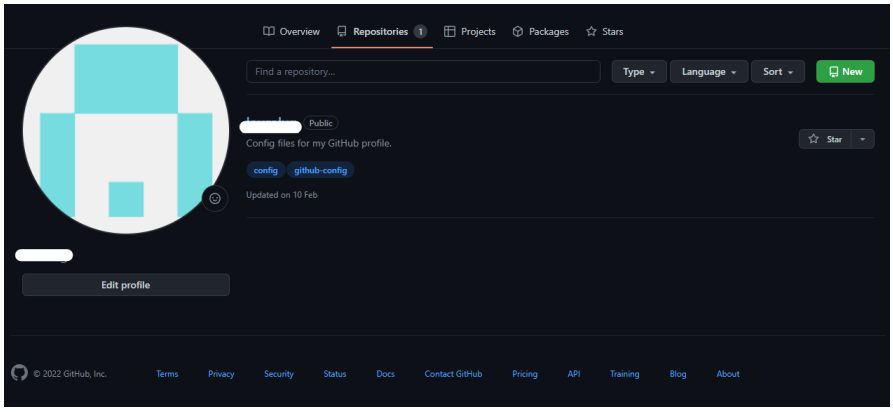

Figura 43. Perfil de GitHub. Interfaz de la aplicación web.

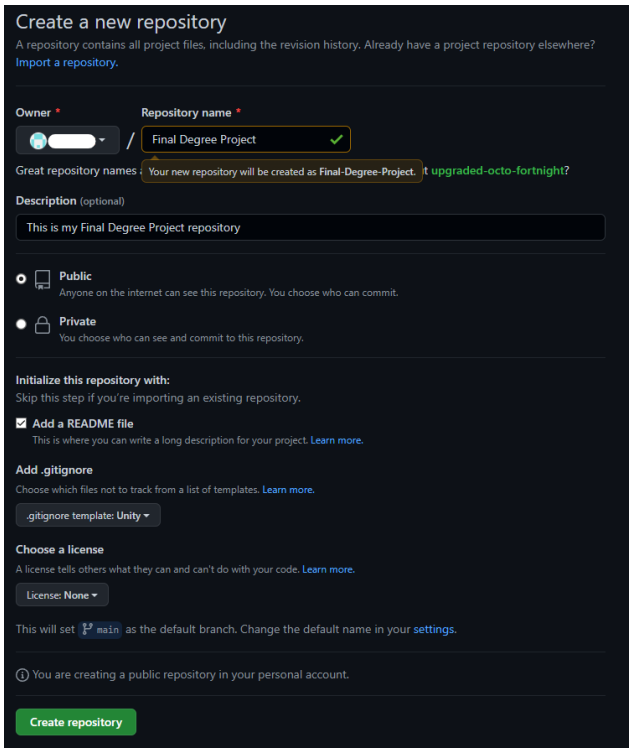

Figura 44. Creación de un repositorio desde la aplicación web.

Sólo es posible crear repositorios públicos si se usa la plataforma de forma gratuita. Se puede, además, incluir un fichero *readme* y otros ficheros más de forma automática: el fichero *gitignore* permite ignorar los archivos cuyas extensiones figuren en dicho fichero. Existe un fichero *gitignore* por defecto para Unity, y es recomendable su uso ya que ignora los archivos temporales utilizados en estos proyectos. Cuando esté creado el repositorio, se deberá subir o crear el proyecto de Unity dentro de éste.

Si se desea, se puede compartir el repositorio a través de un enlace al resto de desarrolladores para que puedan clonarlo según muestra la Figura 47. Esto creará una copia local en su ordenador que podrán editar sin alterar las copias de los otros contribuidores. Para realizar estas tareas de forma más sencilla, es recomendable usar el cliente GitHub Desktop, aunque también se pueden realizar a través de líneas de comando o editando el código desde la aplicación web.

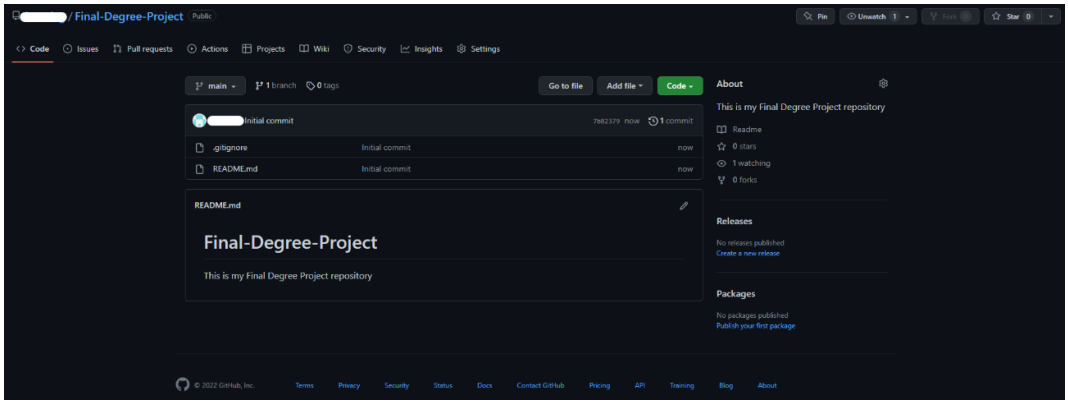

Figura 45. Consulta de un repositorio desde la interfaz de la aplicación web.

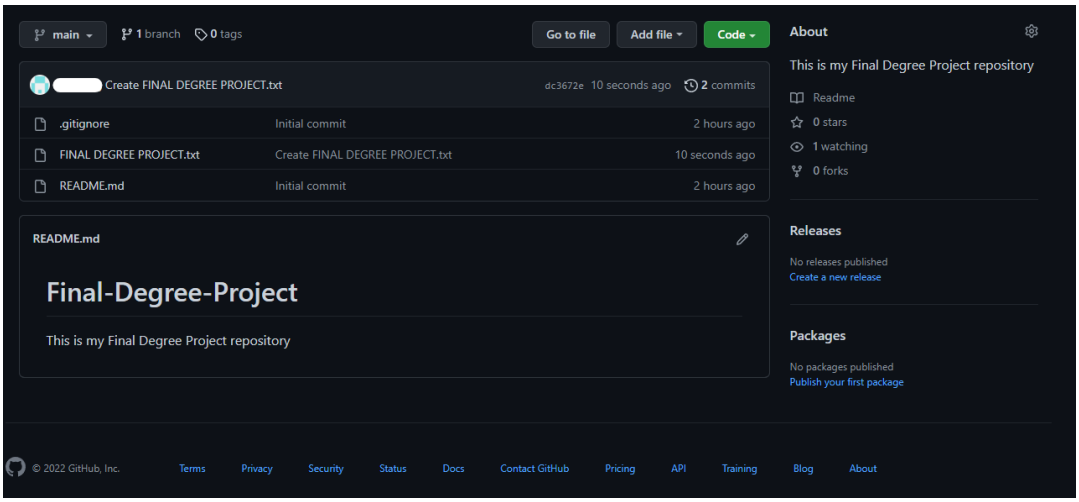

Figura 46. Subida de archivo desde la interfaz de la aplicación web.

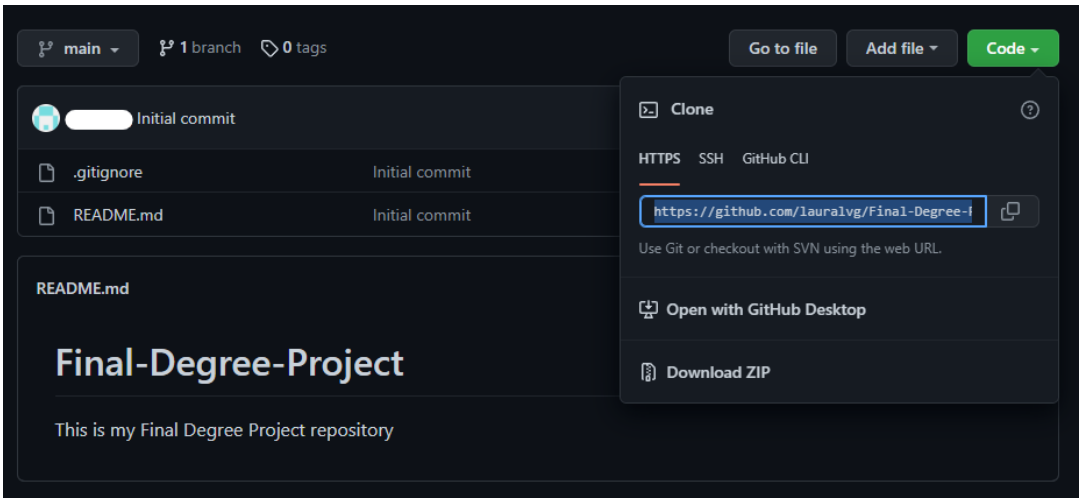

Figura 47. Opciones para compartir el repositorio desde la interfaz de la aplicación web.

Es necesario realizar los siguientes ajustes en Unity para que el control de versiones sea más eficaz, como se muestra en la Figura 48:

- *Edit* > *Project Settings* > *Editor* > *Version Control* > *Visible Meta Files*. Estos archivos guardan información sobre los ajustes y la configuración de cada objeto creado en Unity. Deben estar activados para que sean detectados.
- *Edit* > *Project Settings* > *Editor* > *Asset Serialization* > *Force Text*. Este ajuste hace que muchos de los archivos creados por Unity sean más fáciles de leer.

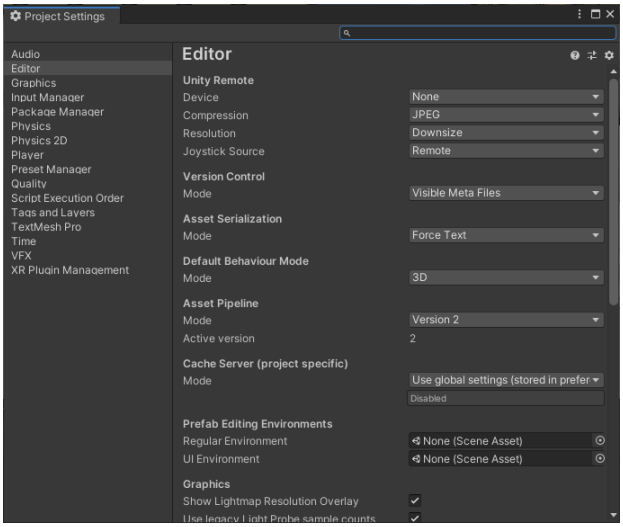

Figura 48. Configuración en Unity para mayor eficacia de uso de GitHub.

Una vez configurado Unity, el repositorio está listo para empezar a añadir cambios al proyecto. Cuando un desarrollador haga cambios en su versión local del proyecto como ocurre en la Figura 49 deberá hacer un *commit*: elegir un título para la actualización que va a hacer y, opcionalmente, una descripción más detallada de dicho cambio. Para que ese cambio se haga efectivo en el proyecto del repositorio, deberá subirlos al repositorio de GitHub, mostrado en la

Figura 50 y denominado *push*. La Figura 51 muestra la interfaz para el desarrollador que ya ha compartido sus cambios.

El resto de los contribuidores recibirán una notificación para actualizar sus copias de proyecto con las modificaciones nuevas, conocido como *pull* y representado en la Figura 52.

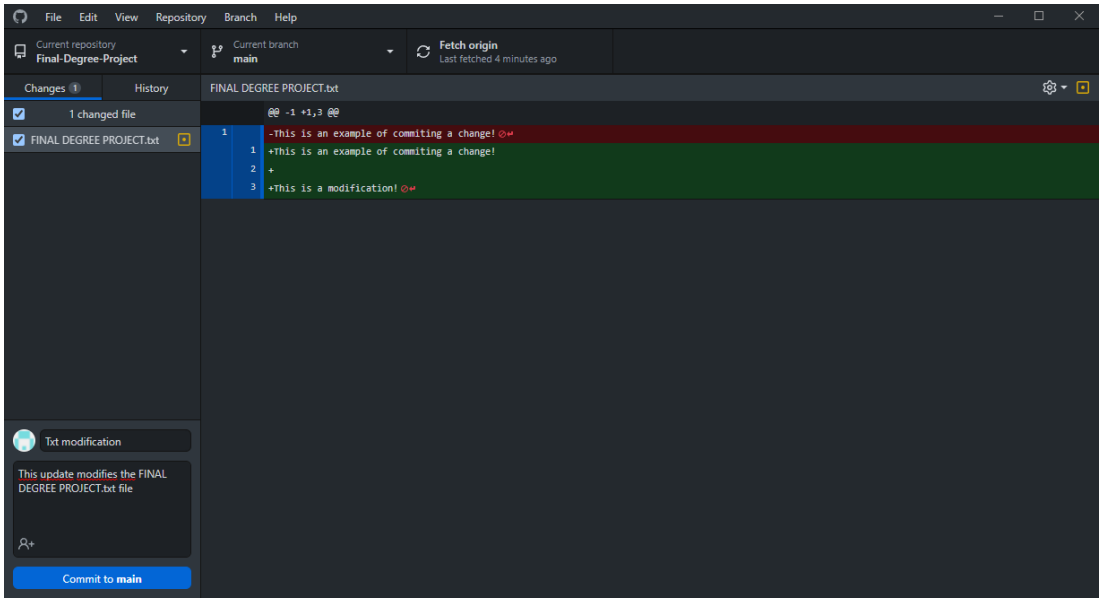

Figura 49. Cambios realizados en el proyecto y la subida de cambios (commit) desde GitHub Desktop.

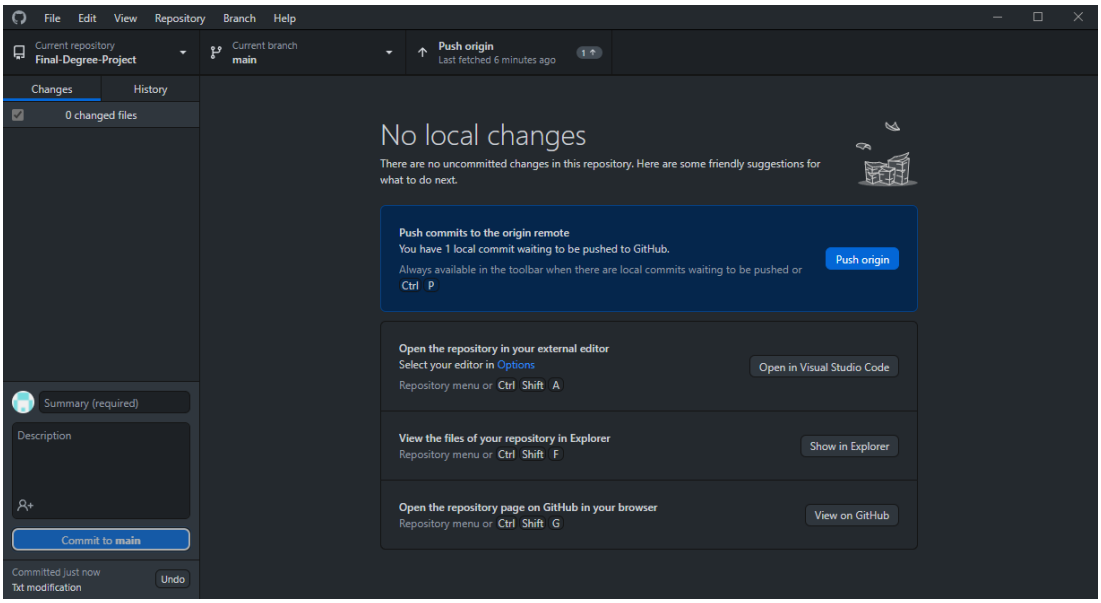

Figura 50. Ejemplo de la acción de subida de un cambio: *push* desde GitHub Desktop.

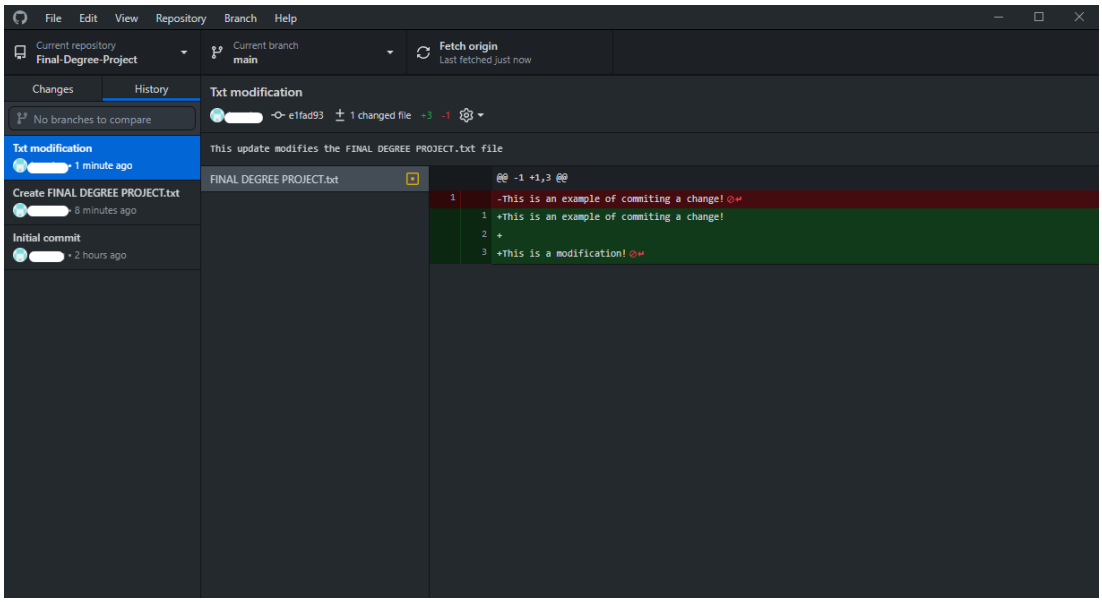

Figura 51. Ventana de historial de cambios desde GitHub Desktop.

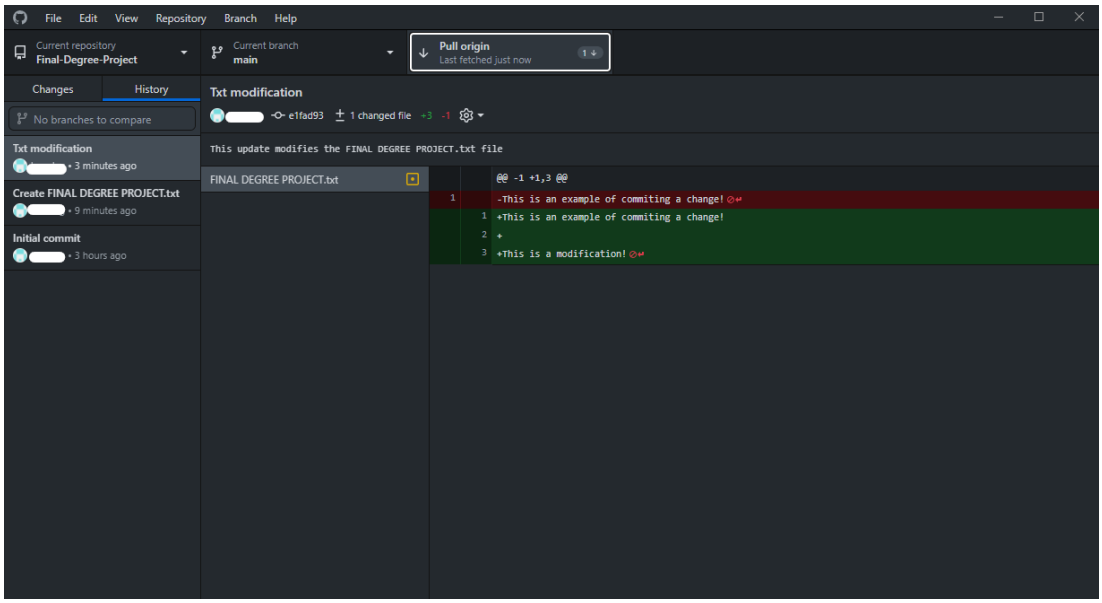

Figura 52. Aviso de descarga de nuevos cambios (*Pull*) desde GitHub Desktop.

GitHub permite trabajar en un mismo repositorio con ramas (*branches*), habiendo siempre una rama principal por defecto denominada *main*. Esto permite tener varias versiones de un mismo proyecto en un solo repositorio. De esta forma y como se puede observar en la Figura 53, cada desarrollador del proyecto puede trabajar de forma independiente en una rama cada uno o subir sus cambios a la rama general.

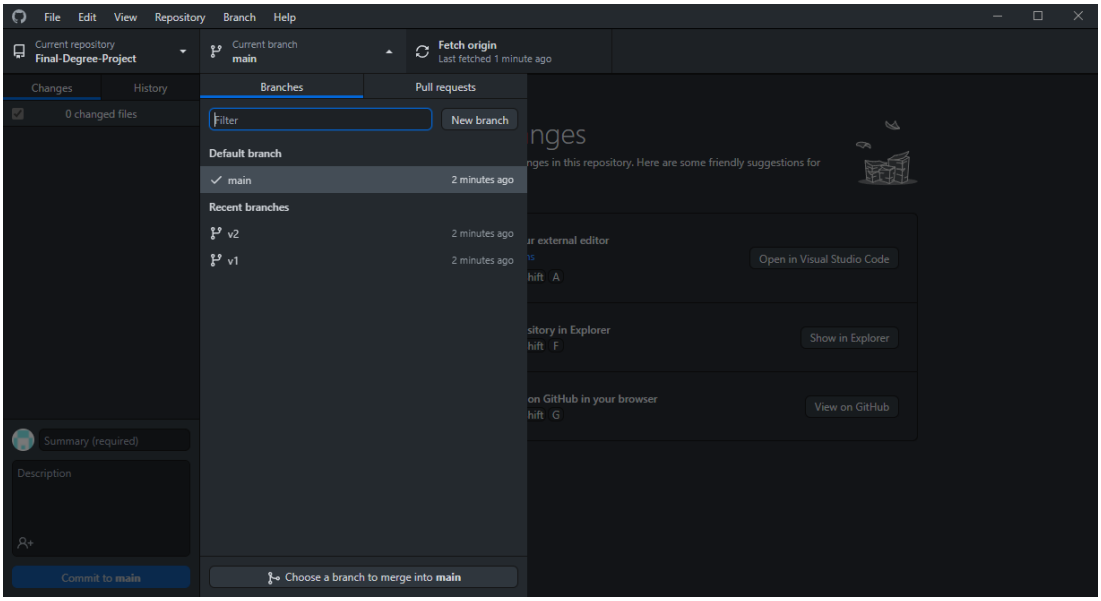

Figura 53. Ejemplo de varias ramas (*branches*) desde GitHub Desktop.

Por último, para manejar proyectos de Unity en GitHub es recomendable el uso de GitLargeFileStorage [1] para el manejo de archivos más pesados que GitHub no puede manejar solo. GitLFS reemplaza aquellos archivos que ocupan más espacio por punteros de texto en Git mientras que los almacena en servidores remotos como GitHub.com o GitHub Enterprise. Sólo es necesario que sea instalado por el dueño del repositorio, como muestra la Figura 54.

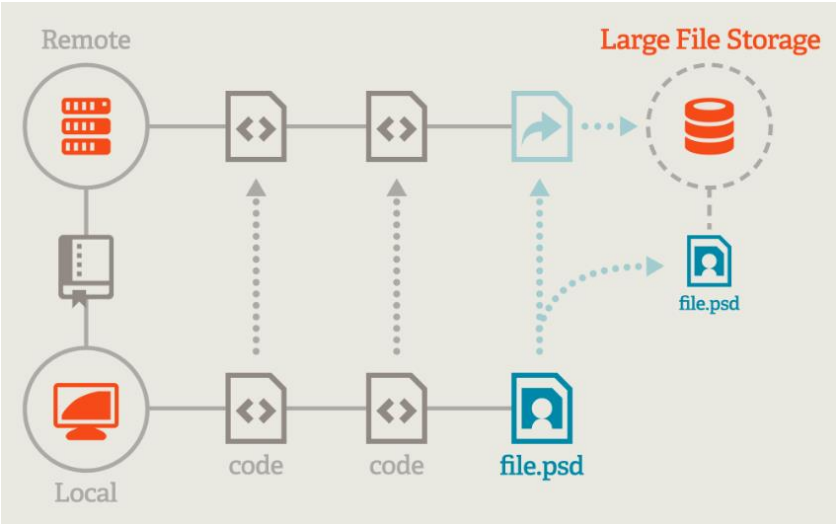

Figura 54. Esquema de funcionamiento de GitLFS.

Cuando ya se ha instalado GitLFS [2] se debe descargar Git [3] para poder configurarlo. Una vez instalado, se debe iniciar Git en el repositorio que se vaya a usar. Como se muestra en

la Figura 55 se deben primero introducir el comando `git lfs start` para comenzar la configuración. Después mediante el comando `git lfs track` se deben poner las extensiones de los archivos que se quieran manejar mediante GitLFs. En el ejemplo se configura para los archivos con extensiones `.mp4` y `.wav`. Por último, se debe usar el comando `git add .gitattributes`. Esto sirve para, en este caso, asignar a esos tipos de archivos el camino necesario para ser referenciado por los punteros de texto.

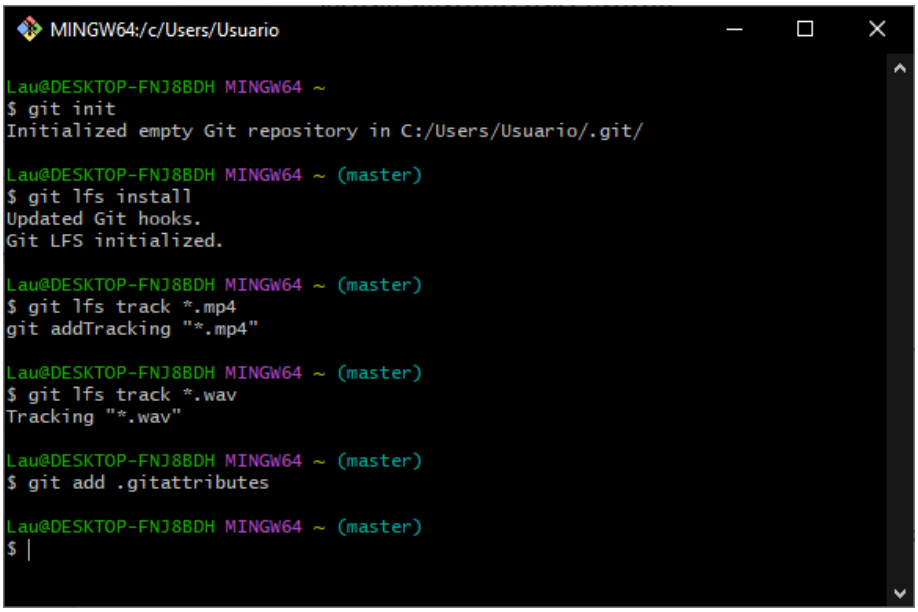

Figura 55. Configuración de GitLFS mediante línea de comandos Git.

Referencias

[1] GitHub, «Instalar Git Large File Storage,» [En línea]. Available: https://docs.github.com/es/repositories/working-with-files/managing-large-files/installing-git-large-file-storage. [Último acceso: 2022 Junio].

[2] Git, «Git Large File Storage,» [En línea]. Available: https://git-lfs.github.com. [Último acceso: Junio 2022].

[3] Git, «Downloads,» [En línea]. Available: https://git-scm.com/downloads. [Último acceso: Junio 2022].