ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Diseño e implementación de una lógica de ajuste terapéutico para el juego serio "*Phibys Adventures 3D*"

AUTOR: Juan Alberto García Fernández

TITULACIÓN: Grado de Ingeniería de Sonido e Imagen

TUTOR: Martina Eckert

DEPARTAMENTO: Teoría de la señal

V°B°

Miembros del Tribunal Calificador:

PRESIDENTE: Julián Nieto Valhondo

SECRETARIO: Enrique Rendón Angulo

Fecha de lectura:

Calificación:

El Secretario,

Agradecimientos

A todos los profesores con los que me he cruzado y me han ayudado durante esta carrera, por los valores que me han inculcado y las enseñanzas que me han transmitido. Especialmente, a mi tutora, Martina Eckert, por la confianza que ha depositado en mi durante la realización de este proyecto.

A mis amigos y amigas, por sus grandes consejos, la tranquilidad, los respiros y la diversión que me han concedido.

Gracias a Maite, por este maravilloso último año de grandes aventuras, amor y calma. Por mostrarme que, tarde o temprano, se hace el camino.

Y por supuesto, quiero agradecer a mis padres y a mi hermana, que han sido un grandísimo apoyo desde casa durante los años como estudiante en esta escuela y durante todas las etapas de mi vida.

Resumen

Este Proyecto de Fin de Grado versa en torno al diseño y la implementación de una lógica de ajuste terapéutico para un juego serio titulado *Phiby´s Adventure 3D*, un juego para la rehabilitación física de personas con discapacidad, basado en la primera versión llamada *Phiby´s Adventure* y de la cual se han extraído algunos elementos principales.

Asimismo, este proyecto tiene como objetivo principal la fundación de una arquitectura estable del videojuego, así como el diseño de las diferentes componentes que conforman el entorno, los cuales son útiles para la generación de la lógica principal del juego. Por otro lado, al margen de estos elementos presentes en el entorno, se precisa del establecimiento de niveles de dificultad mediante el uso de parámetros que se pueden configurar a distancia desde la plataforma web Blexer-Med, que fue desarrollada para la versión anterior del juego. La comunicación entre la web, el juego y el sensor óptico Microsoft Kinect 2.0 que captura los movimientos del jugador en los que se basa el juego, la realiza un *middleware* llamado K2UM (*Kinect to Unity Middleware*) que fue desarrollado y ampliado por otros estudiantes anteriormente y durante este proyecto.

Con el fin de poner en funcionamiento el desarrollo de una solución, ha sido indispensable la integración de diversas herramientas obtenidas de múltiples proyectos anteriores, las cuales han sido beneficiosas para la evolución del videojuego. Asimismo, durante este periodo, se ha creado una organización jerárquica para el establecimiento de un trabajo eficiente y se han instaurado unas bases de integración a fin de la creación de un proyecto estable que puede servir como base para el desarrollo continuo.

Por otro lado, en base a las escenas ya creadas e integradas en el proyecto, se ha generado el ajuste entre cambios de escena, el cual ha sido el desencadenante de la integración del nuevo *Kinect Asset*, que ha servido para la exportación de parámetros de la web que, mediante diversas funciones algorítmicas, han sido utilizables para la generación de un juego un poco más versátil. De esta forma, a partir de los parámetros extraídos de la web, se ha creado una base para un sistema de integrar ejercicios terapéuticos en el juego.

A lo largo de la elaboración de este Proyecto de Fin de Grado, se ha conseguido fundar la base para un juego de gran magnitud. Con él, se ha posibilitado marcar una dirección para futuras líneas de investigación a partir de los resultados alcanzados actualmente.

Abstract

This Final Degree Project is about the design and implementation of a therapeutic adjustment logic for a serious game entitled *Phiby´s Adventure 3D*, a game based on the first version called *Phiby´s Adventure* and from which some main elements have been extracted.

Likewise, this project has as its main objective the foundation of a stable videogame architecture, as well as the design of different components that make up the environment, which are useful for generating the main logic of the game. On the other hand, apart from these elements present in the environment, it is necessary to establish levels of difficulty through the use of parameters that can be configured remotely from the Blexer-Med web platform, which was developed for the previous version of the game. The communication between the web, the game and the Microsoft Kinect 2.0 optical sensor (which captures the movements of the player on which the game is based) is done by a *middleware* called K2UM (*Kinect to Unity Middleware*), that was developed and expanded by other students before and during this project. In order to put into operation the development of a solution, it has been essential to integrate various tools obtained from previous projects, which have been beneficial for the evolution of the videogame. Also, during this period, a hierarchical organization has been created for the establishment of efficient work and integration bases have been established, creating a stable project.

On the other hand, based on the scenes already created and integrated in the project, the adjustment between scene changes has been generated, which has been the trigger for the integration of the new *Kinect Asset*, that has been used to export parameters of the web that, by means of diverse algorithmic functions, have been usable for the generation of a more versatile game. In this way, based on the parameters extracted from the web, the therapeutic exercise system has been created. Like this, the creation of a template that replaces the origin of these parameters is required if the configuration provided by the web is not used.

Throughout the elaboration of this Final Degree Project, it has been possible to build the base of a game of great magnitude. With it, it has been possible to set a direction for future research lines based on the results currently achieved.

Índice de contenido

Índice de figuras

Lista de acrónimos

GAMMA: Grupo de Aplicaciones Multimedia y Acústica.

CITSEM: Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad.

K2UM: *Kinect to Unity Middleware*.

NPC: Personaje no jugador (*Non-player Characters*).

FPC: Juego en primera persona (*First Person Controller*).

TPC: Controlador en Tercera Persona (*Third Person Controller*).

UI: Interfaz de Usuario (*User Interface*).

PFG: Proyecto de Fin de Grado.

RV: Realidad Virtual (*Virtual Reality*).

1. Introducción

En los últimos años se han realizado múltiples avances en la investigación del desarrollo de nuevas tecnologías en terapia para personas con movilidad reducida . Una de ellas es la realidad aumentada, la cual se ha ramificado gracias a diferentes estudios.

De esta manera, desde el grupo de Aplicaciones Multimedia y Acústica (GAMMA) del Centro de Investigación en Tecnologías del Software y Sistemas Multimedia para la Sostenibilidad (CITSEM) se están desarrollando diversas aplicaciones para personas con diversidad funcional motora, principalmente niños, con fines terapéuticos basados en el movimiento corporal.

Para ello, gracias al dispositivo creado por la compañía Microsoft denominado Kinect 2.0 [1], se procede al desarrollo de varios videojuegos. Entre ellos, en 2014 comienza *Phibys Adventures v1*, realizado con el programa Blender [2]. En 2018 se realiza la demo de *Phiby's Adventures 3D* y un nuevo *middleware* K2UM [3] que proporciona la comunicación entre la Kinect 2.0 y el programa Unity 3D [4].

Posteriormente, se elabora la segunda versión de *Phiby's Adventures* en Unity 3D por un equipo de estudiantes de PFG y prácticas que cambia cada semestre. Dentro del equipo, este Proyecto Fin de Grado (PFG) se enmarca en:

1. El desarrollo, actualización y cambio entre diferentes escenas
2. La implementación de objetos funcionales dinámicos en Unity 3D
3. El diseño de una arquitectura de juego estable, la comunicación entre una plataforma web y el juego para el ajuste de la dificultad de niveles según las capacidades del paciente.
4. La generación de parámetros por defecto para la independencia de este de la web.

Para realizar el proyecto se ha requerido un estudio previo de la plataforma web médica Blexer-Med [5] y el *middleware* K2UM 2.0.

1.1. Objetivos

Entre los objetivos principales para este PFG se encuentran los siguientes:

1. Planificación, diseño y creación de la arquitectura general del contenido del juego:
   - Como parte de un equipo de estudiantes, acordando así las mecánicas de juego, para conseguir un funcionamiento fluido y estable de un prototipo del juego.
   - Diseño de una estructura de proyecto genérica y manejable.
   - Conseguir la integración de parámetros entre proyectos heredados.
2. Diseño de una lógica de ajuste terapéutico que consiste en el establecimiento de una nomenclatura para un ejercicio y la fijación de parámetros ajustables por ejercicio a través de una web Blexer-Med por el terapeuta, tales como:
   - Número de veces que se realiza un movimiento (entre los valores mínimo y máximo).
   - Tiempo máximo para terminar la escena.
   - Frecuencia de apertura y cierre de escenas y en el mapa.
3. Diseño de diferentes elementos prácticos y creativos para una futura línea de trabajo, creando de esta manera una mejora en la experiencia del juego para el usuario. De esta forma, se consigue una mayor funcionalidad de juego, de tal forma que se practiquen ejercicios terapéuticos, no sólo en una escena concreta, sino además durante la estancia en el mundo abierto. Se pretende que las mecánicas utilizadas en el juego sean útiles para, a través de ejercicios diversos, se consiga una mayor interacción sin que el jugador acabe fatigado.

1.2. Especificaciones y restricciones del diseño

Las especificaciones del diseño previo a una solución desarrollada vienen dadas por las siguientes herramientas, las cuales aportarán una gran fuente de ventajas y facilitarán el avance del proyecto. Por contraste, estas herramientas también comprenden una serie de limitaciones que condicionarán la obtención de una solución completa y depurada, lo cual lleva a una constante mejora y actualización en el futuro. Entre estas tecnologías se encuentran:

- Motor de videojuegos de Unity 3D.
- Cámara Microsoft Kinect 2.0 para la captura de movimientos en tres dimensiones. Puede haber pequeños inconvenientes en la precisión de la captura de todas las articulaciones.
- Programa *Kinect 2 Unity Middleware* (K2UM 2.0) el cual puede requerir de nuevas exigencias para el juego.
- *Visual Studio* como editor para la programación de *Scripts*.
- Programación C#, el lenguaje más popular utilizado para Unity 3D.
- Elementos de integración basados en proyectos creados con anterioridad, como escenas, *Scripts*, objetos y otras componentes, de los cuales algunos cambios podrán ser requeridos para la solución de este proyecto.
- Web de Blexer-Med para el empleo de datos en esta.
- *Kinect Asset* para la comunicación con la Kinect 2.0 y con la web Blexer-Med.

2. Marco tecnológico

2.1. Estado del arte

En la actualidad, y tras el gran avance de las nuevas tecnologías en las últimas décadas, están presentes una gran variedad de campos de investigación dedicados a la rehabilitación de pacientes con diversidad funcional física. A día de hoy, gracias a la realidad virtual y la realidad aumentada, existen una gran cantidad de entornos especializados en este campo que sirven para convertir los hábitos de tratamientos monótonos en experiencias dinámicas e interactivas en las que se consigue una mayor inmersión del paciente en el entorno.

Entre otras, una plataforma web especializada en la neurorrehabilitación y estimulación cognitiva es *NeuronUP* [6], una herramienta muy útil para profesionales en neurorrehabilitación, con ejercicios para Alzheimer, Parkinson, Esclerosis Múltiple, Daño cerebral, Ictus o TDAH entre otros (Figura 1).

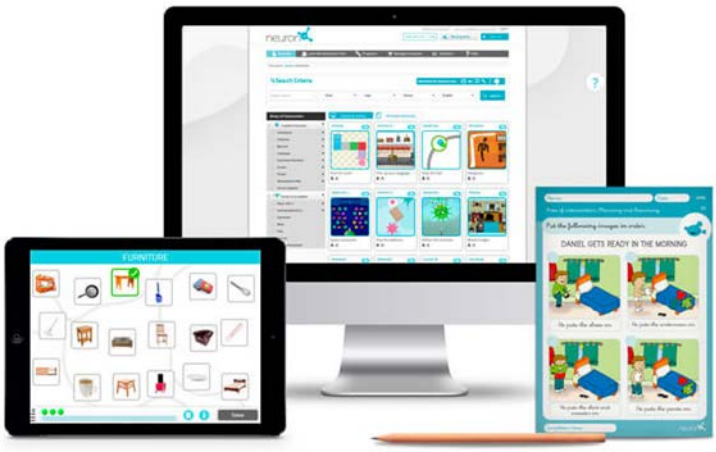


Figura 1. Aplicación de NeuronUP

Esta plataforma se utiliza por diversos centros, entre ellos, por lo centros de la *Red Menni* de *Hermanas Hospitalarias* [7], la cual acude a otras tecnologías como la cinta de marcha *Hp Cosmos*, con el sistema de fijaciones elásticas *RoboWalk*® [8]. Véase en la Figura 2.

Figura 2. Centro *Red Menni* (izquierda) *y RoboWalk*® de *Hp Cosmos* (derecha)

Por otra parte, el centro biomecánico *Umana* presenta una innovadora tecnología de RV con aplicación a la terapia psicomotriz para pacientes con lesiones, dificultades biomecánicas, con daño cerebral y/o medular, niños con disfunciones y personas de la tercera edad. Esta tecnología, llamada *vinCi,* consiste en capturar el movimiento del paciente e inmediatamente trasladarlo este al entorno virtual. Este tipo de tecnologías permiten amenizar las rutinas de entrenamiento mediante juegos [9]. Véase en la Figura 3.

Figura 3. Paciente realizando ejercicios con *vinCi*

Asimismo, este tipo de tecnologías permiten al personal sanitario almacenar los resultados obtenidos del paciente en una base de datos y comparar los resultados obtenidos durante el transcurso de la terapia y así ser capaz de evaluar el progreso de este. De la misma manera, el terapeuta puede controlar los eventos que sucedan en tiempo real durante el transcurso del juego (Figura 4).

Figura 4. Terapeuta ajustando la configuración de los ejercicios con VinCi

Un objetivo semejante a este tipo de tecnologías es el que se lleva a cabo en el proyecto *Blexer*, desarrollado desde 2015 por el grupo GAMMA, en el que la propuesta principal es la creación de un entorno virtual en el que se puedan llevar a la práctica una serie de ejercicios propuestos para personas con diversidad funcional motora.

2.2. Marco tecnológico para el desarrollo de la solución

2.2.1. Microsoft Kinect 2.0

Se trata de un controlador de juego libre y entretenimiento desarrollado por la empresa Microsoft para la videoconsola Xbox 360 y, posteriormente, tras la aparición de Kinect 2.0 (véase en la Figura 5), para la Xbox One. Además, a partir de junio de 2011, la Kinect comenzó a ser compatible para PC a través de Windows 7 y Windows 8.

Figura 5. *Hardware* de la Kinect 2.0

Características de la Microsoft Kinect 2.0:

1. Gran campo de visión (0º en horizontal y 60 en vertical) y resolución 1920 x 1080 Full HD
2. Sensor de profundidad de alto rango (4,5 metros)
3. USB 3.0
4. Avanzada captación de sonidos (mejora en el reconocimiento de voz con respecto a la 1.0)
5. Captación de movimientos en la oscuridad (Kinect 2.0)
6. Cálculo de la fuerza muscular y la medición del ritmo cardiaco.
7. Detección de 6 personas de forma simultánea.

2.2.2. Unity 3D

Unity 3D es un motor de videojuego multiplataforma creado por la compañía Unity Technologies y disponible para Microsoft Windows, OS X y Linux. Este motor se puede utilizar junto con otros programas, principalmente de diseño 3D, como son Blender, 3ds Max, Maya, Softimage, Modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks y Allegorithmic Substance [10].

Actualmente Unity 3D [11] es una de las plataformas de diseño para videojuegos más famosas, principalmente por dos razones: se trata de una plataforma gratuita para uso personal o para uso comercial con unos ingresos o fondos de menos de 100.000 $ en los últimos 12 meses. Véase un ejemplo del entorno Unity 3D en la Figura 6.

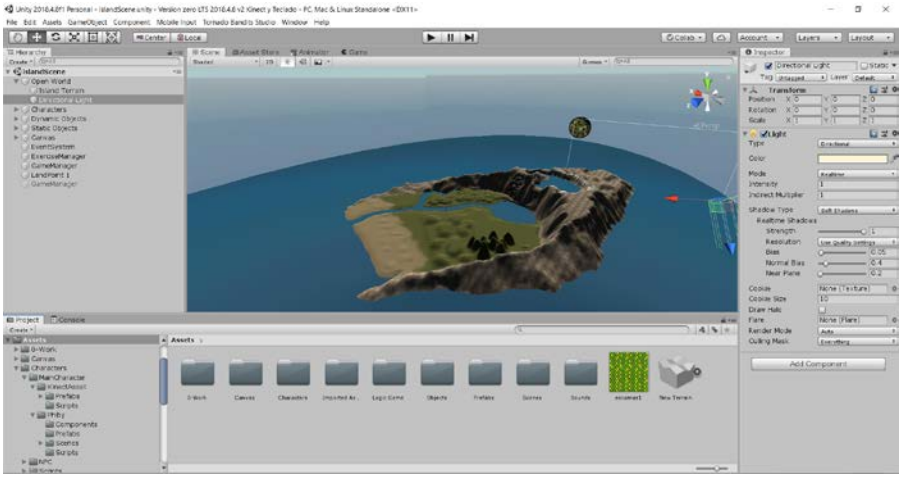


Figura 6. Captura del entorno Unity 3D

Características [12]:

- Los leguajes de programación principales que ha utilizado Unity 3D para el desarrollo son Boo (actualmente obsoleto), Unity *Script* (una versión particular de JavaScript) y C# (el lenguaje más común utilizado para esta plataforma).
- Permite la importación de modelos y animaciones 3D desde otros programas, como los mencionados anteriormente.
- A través del *Inspector* se pueden controlar una gran cantidad de características tanto a nivel de Objeto 3D como a nivel de físicas de este objeto y del entorno que le rodea. Además, se pueden crear y realizar cambios en la iluminación de los diferentes espacios del mundo 3D.

Durante el proyecto, se ha optado por este motor, no sólo gracias a las características que dispone, sino también por las posibilidades y funcionalidades que ofrece. Es un motor creado para hacer videojuegos profesionales, el cual no requiere de ningún pago inicial, esto es una ventaja a nivel competitivo con otras empresas, además de serlo en materias de investigación. Además, es un motor versátil, el cual está comunicado con otros programas de diseño 3D profesionales, como los mencionados anteriormente.

Por otra parte, se trata de un motor utilizado en asignaturas en esta escuela, por lo que, a nivel estudiantil, es útil para los alumnos que participen en este gran proyecto.

2.3. Antecedentes del juego en desarrollo

2.3.1. Phibys adventures v1

Durante las primeras fases, se creó la versión inicial de *Phiby´s Adventure* en el programa Blender, donde el proyecto está dividido en dos facciones diferenciadas: el modelado 3D de las partes implicadas en el juego y la creación de una lógica de juego que permite una ejecución combinada de código mediante *scripts* en el lenguaje *Phyton*. La mecánica principal de los ejercicios se sustentó en los movimientos capturados por la cámara Microsoft Kinect X360 [13]. El propósito de este proyecto era conectar cuatro ejercicios básicos de forma lúdica, para que el jugador los repitiera múltiples veces durante el juego, con el objetivo de ganar el juego sin percatarse de que este realizando estos ejercicios.

Los cuatro ejercicios, mostrados en la Figura 7, se combinan entrelazando al personaje principal Phiby en una serie de escenas donde el jugador puede elegir a qué lugar de cada escena quiere avanzar y, por lo tanto, qué ejercicio desea realizar. Mediante un mapa restringido con todas las escenas en forma de celdas, Phiby continúa su aventura llevando a la práctica los diferentes ejercicios. La acotación de este mapa es requerida, dado que se vio limitada por la cantidad de recursos a cargar en tiempo real por el motor gráfico de Blender.

En el origen, los cuatro juegos principales que se desarrollaron en este programa fueron: *"Row the boat"*, *"Chop the Wood"*, *"Dive and eat"* y "*Climb the tree"*.

- *Row the boat*: consiste en remar el número de metros ajustados por el terapeuta, con un movimiento simultaneo de los brazos en un tiempo limitado por el este.
- *Chop the Wood*: en este ejercicio, el objetivo es cortar el número de troncos ajustados por el terapeuta en el tiempo limitado por él. Se levanta el brazo a la espera de que se "cargue" el hacha, visualizado por una estrella. En ese momento se baja el brazo para cortar el tronco.
- *Dive and eat*: este ejercicio consiste en recolectar el mayor número de plancton en un tiempo limitado. El manejo del personaje se realiza mediante la rotación del tronco de izquierda a derecha y de arriba abajo.
- *Climb the tree:* mediante el movimiento alterno de los brazos de arriba abajo, el objetivo del jugador es trepar el un número de metros en el árbol hacia arriba, también en un tiempo límite.

Figura 7. Escenas de *Phiby´s Adventure v1*

Para la coordinación de las diferentes partes del diagrama mostrado en la Figura 8 y teniendo en cuenta que Blender no permite una comunicación directa con la Kinect X360 se requiere del desarrollo de un *middleware* que sirva como herramienta de comunicación entre el entorno y otros dispositivos. De esta forma, se procede al desarrollo e implementación del *middleware* Chiro por parte de Ignacio Gómez-Martinho [14] que permite la conexión con la web Blexer-Med v1.

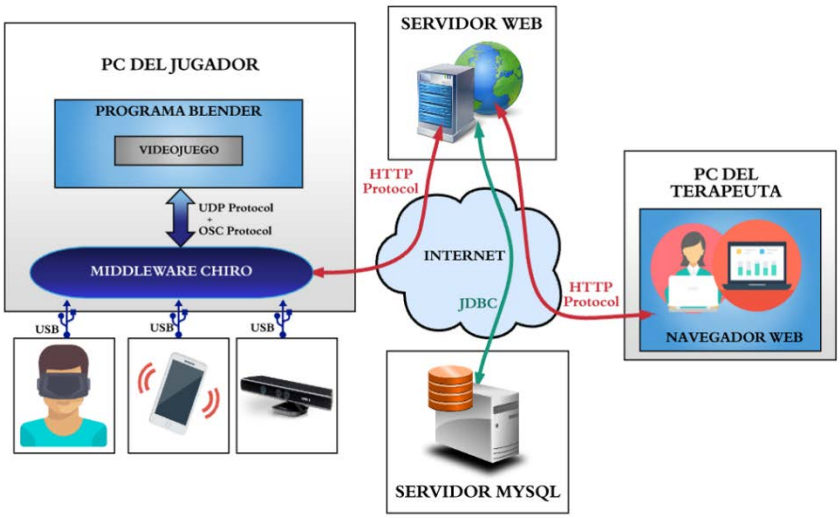


Figura 8. Comunicación de los diferentes elementos con el K2UM [2]

Paralelamente nace la primera versión de la web Blexer-Med (Figura 9), creada por Mónica Jiménez Ramos [15], que sirve como instrumento de comunicación entre el terapeuta y el videojuego. De esta manera, el terapeuta es capaz de adaptar los diferentes niveles de dificultad a la necesidad de cada paciente de forma particular y se podrán guardar y revisar los datos de los ejercicios realizados por los pacientes.

Figura 9. Página de bienvenida de Blexer-Med y autenticación para terapeutas [15]

2.3.2. Kinect to Unity Middleware

K2UM 1.0

Mediante el proyecto realizado por César Luaces [3], se crea la primera versión de un *middleware* que sirve para la gestión de la comunicación con la Microsoft Kinect 2.0. A través del envío de las articulaciones activas, accede a la información de movimiento proporcionada por vía USB y procesa estos datos para la comunicación con el motor de videojuegos Unity 3D como se puede ver en la Figura 10.

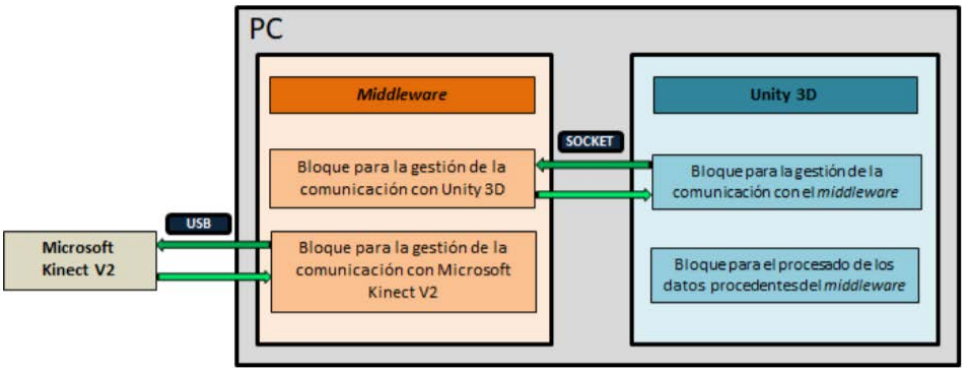


Figura 10. Diagrama de bloques general del K2UM [2]

A continuación, se detalla de forma breve el funcionamiento de la comunicación directa entre el *middleware* K2UM y la aplicación de Unity 3D.

El contenido de los diferentes campos de envío y recepción de datos es el siguiente:

- Por un lado, están los tipos, que manifiestan si se trata de mensajes de datos y mensajes de control.
- Dentro de cada mensaje de datos y de cada mensaje de control, estos se subdividen en diferentes clases, las cuales se agregan entre paréntesis al lado del nombre de la clase.

Mensajes de datos:

Se encuentran dos clases de mensajes de datos, los **datos de posición (PP)**, que contienen la información de las tres coordenadas de espaciales para cada articulación, con el nombre de esta y los **datos de rotación (QQ),** los cuales contienen la información de las cuatro variables de rotación para cada articulación (cuaterniones), con el nombre de este.

Mensajes de control:

- **Información de articulaciones activas (*AJ*):** esta información se envía desde la aplicación al *middleware* una vez al inicio de la conexión con el juego. Esta incluye una cadena de caracteres con las articulaciones activas (*Active Joins*) del objeto 3D utilizado.
- **Solicitud de datos de movimiento (*SP*):** esta información es transmitida desde el *middleware* y sirve para solicitar un mensaje de datos al juego. El mensaje de datos que desee solicitar puede ser de tipo Q (Solicitud de datos de clase "Datos de rotación"), tipo P (Solicitud de datos de clase "Datos de posición") y tipo B (Solicitud de datos de ambas clases).
- **Desconexión desde la aplicación (*PM*):** Se envía desde Unity 3D al *middleware* para avisarle de que ha terminado su ejecución. En ese mismo instante, el *middleware* se queda en escucha para la nueva recepción de información de control.
- **Nombre de usuario (*UN*):** Mensaje enviado desde la aplicación al *middleware* con el nombre del usuario.
- **Resultados de ejecución (*FR*):** Este mensaje se envía desde la aplicación al *middleware* para informar sobre los resultados obtenidos.
- **Mensajes de error (*ER*):** Desde el K2UM a la aplicación se envía un mensaje informando de un error que debe ser resuelto antes de continuar con la ejecución. Existen cuatro tipos de mensajes de error que vienen dados en la cadena de caracteres:
    - ***AJ**:* No ha sido posible procesar el mensaje de articulaciones activas recibido. Se solicita un nuevo envío de estas por parte de la aplicación.
    - ***SP**:* No ha sido posible procesar el mensaje de solicitud de datos de movimiento. Se solicita un nuevo envío de estos por parte de la aplicación.
    - ***NT**:* No se ha encontrado la localización de ningún usuario de forma correcta desde la Kinect 2.0. Previamente se notifica este mensaje desde la *Kinect* al *Middleware*.
    - ***AT**:* Este mensaje es enviado posteriormente al mensaje de error ***NT***, anunciando a la aplicación que se ha vuelto a recuperar la localización del usuario.

A continuación, en la Figura 11 se muestra el esquema general del proceso de envío y recepción de mensajes entre el *middleware* y la aplicación de Unity 3D para la comunicación entre ellos. Primero se envían los mensajes de control de articulaciones activas, el nombre de usuario y el de solicitud de datos de movimiento (posición y rotación) desde la aplicación. A continuación, el *middleware* hace un envío de los

mensajes de datos de posición y rotación solicitados. Una vez recibidos por la aplicación, esta sigue solicitando datos de movimiento y recibiendo repetidamente estos por el *middleware*. Al final del ejercicio, la aplicación envía un mensaje de resultados del ejercicio y otro de desconexión desde la aplicación. En el transcurso de este periodo de comunicación, se utilizan los mensajes de error en caso de que se haya perdido cierta información en el recorrido.

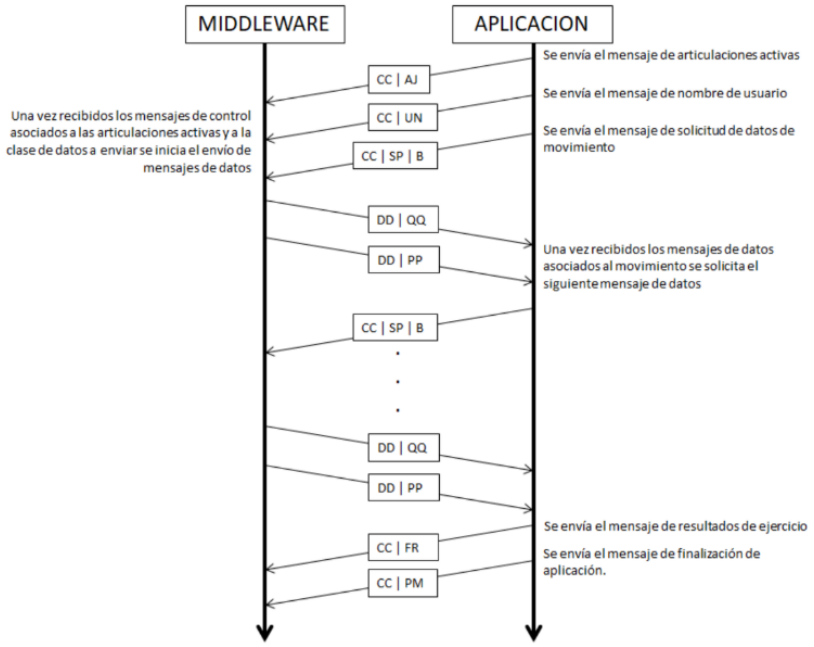


Figura 11. Esquema de transmisión esperado en una comunicación entre el *middleware* y la aplicación [3]

Es necesario entender este proceso de transmisión de datos para poder saber qué está sucediendo en cada instante de tiempo durante la comunicación. Gracias a este estudio previo, se evitan contratiempos en el traslado de componentes.

K2UM 2.0

En 2019 se actualizó el *middleware* K2UM para la comunicación con la web Blexer-Med, como se muestra en el diagrama de la Figura 12. Además, en este proceso se procede a la implementación de la visualización de la cámara y el esqueleto de la Kinect 2.0. Todo este proceso es llevado a cabo por Daniel Iglesias [16] durante la Beca de Colaboración con el departamento.

Como se puede ver, el primer paso es la autenticación del usuario a través del **LoginForm.cs**. En caso de que el usuario esté registrado como paciente, éste deberá introducir su nombre de usuario y contraseña, en caso contrario, tendrá que continuar sin cargar la configuración. Además, se han implementado también una serie de excepciones en caso de no rellenar los datos o rellenarlos de forma inadecuada.

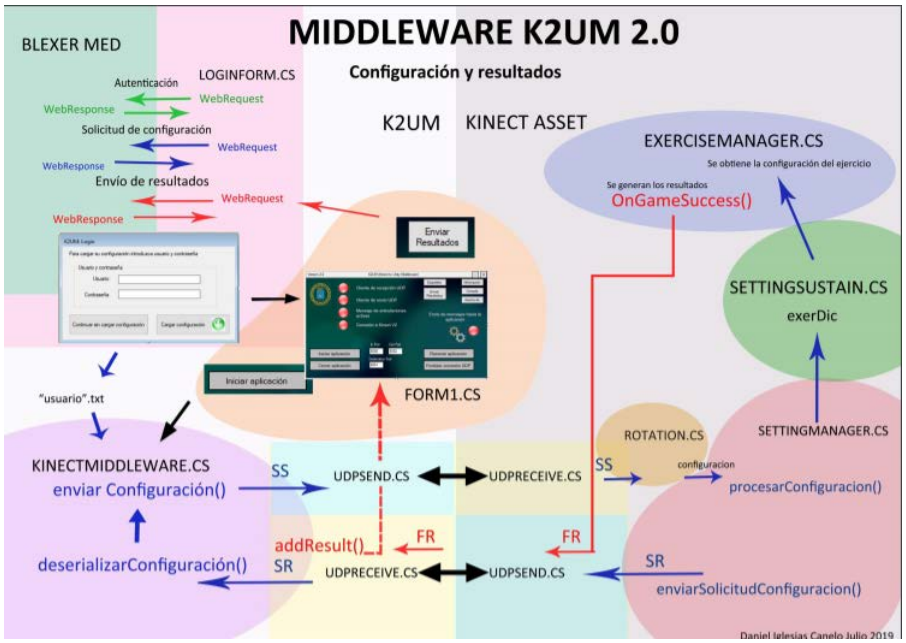


Figura 12. *Middleware* K2UM 2.0 [16]

Tras haberse descargado los datos de configuración de la web, se inicia el proceso de conexión con la Kinect 2.0. Además, se integran los nuevos *Scripts* del *Kinect Asset* al videojuego para configurar éste de forma adecuada.

El **SettingManager.cs** es el encargado del segundo paso en el nuevo proceso de conexión entre Unity 3D y el *middleware*, que consiste en el envío del mensaje de control cada 100 *frames* con el formato que se muestra en la Figura 13 hasta que reciba los datos del usuario. Este proceso es básico para comunicar al *middleware* cuál es el juego que se está ejecutando.

K2UM	CC	####	SR	"[gameId, gameCode, gameTitle]"
Cabecera	Tipo	Longitud Datos	Clase	Datos
8 bytes	2 bytes	4 bytes	2 bytes	N bytes

Figura 13. Mensaje de control generado por el SetingManager.cs [16]

Una vez el *middleware* recibe este mensaje de solicitud, se inicia el proceso de deserialización de para el procesado de los datos del videojuego, mediante el método **deserializarJuegoSolicitado()** del *Script* **KinectMiddleware.cs**.

Después de haber recibido la información de los datos de configuración del videojuego, el K2UM envía un mensaje mediante el protocolo UDP a Unity 3D con el formato mostrado en la Figura 14 serializando primero la variable *loadSettingResponse* en una cadena de texto mediante el método **serializarConfiguración()** y enviándolo gracias al método **enviarConfiguracion()** a través de una cola de mensajes.

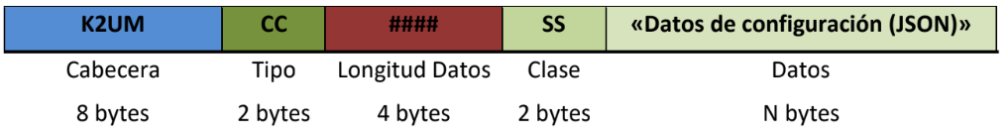


Figura 14. Mensaje de configuración por parte del K2UM 2.0 [16]

Tras estos pasos, el *Script* **UDPReceive.cs** del *Kinect Asset* de Unity 3D capta el mensaje recibido, el cual remite al *Script* **Rotation.cs**, que se encarga de guardar estos datos en la variable configuración de tipo *String*, la cual puede tomar tres estados:

- Vacía, cuando no se ha recibido ningún mensaje de configuración (cadena de caracteres: “”).
- Nula, cuando el usuario ha marcado la opción de cargar sin configuración (cadena de caracteres: “null”)
- Con datos de configuración, en el caso de que se hayan cargado los datos de configuración (cadena de caracteres: “Datos de configuración (JSON)”).

En el final de este proceso, el **exerciseManager.cs** asociado al objeto *Exercise Manager*, accede a la componente *Script* **SettingSustain.cs** asociada al objeto *Setting Objetct*, de la cual se obtienen los parámetros de la web para un ejercicio concreto. Estas dos componentes, junto con el **settingManager.cs**, sirven para el correcto enlace entre el juego y la web, por lo que se tratan de elementos fundamentales para el desarrollo de la solución de este PFG. De este modo, se hablará de su función y aplicación en el proyecto en apartados posteriores.

No obstante, para obtener estos parámetros, se deben introducir antes los parámetros “Identificador de juego”, “Código de juego” y “Nombre de Juego” en el **exerciseManager.cs** de Unity 3D. Adicionalmente, se pueden enviar los resultados obtenidos en el juego a la web.

Posteriormente se hará más hincapié en esta segunda versión del K2UM, pues entra en contacto directo con las soluciones proporcionadas por este PFG.

K2UM 3.0

En un proyecto paralélelo, en el cual Gerardo Cilleruelo creó el juego terapéutico WestGun-Therapy [17], se modificó el *middleware* para introducir la funcionalidad de poder mover el cursor de la pantalla con el movimiento de la mano. En este proyecto, se captura el movimiento del brazo en tiempo real y se replica este en el entorno virtual. Además, este movimiento tiene un sistema de amplificación el cual sirve para pacientes que presenten dificultades en el movimiento del brazo. De esta forma, se consigue un movimiento más amplio a nivel virtual, con acciones que requieran menos esfuerzo.

Como se puede ver en el diagrama de la Figura 15, durante la calibración antes del comienzo del juego, se le asigna el cursor a la zona del esqueleto del brazo. Este proceso se realiza gracias a la comunicación entre Unity 3D y el *middleware* y la transformación del espacio en 3D recibido a través de la Kinect al espacio 2D ocupado en la pantalla.

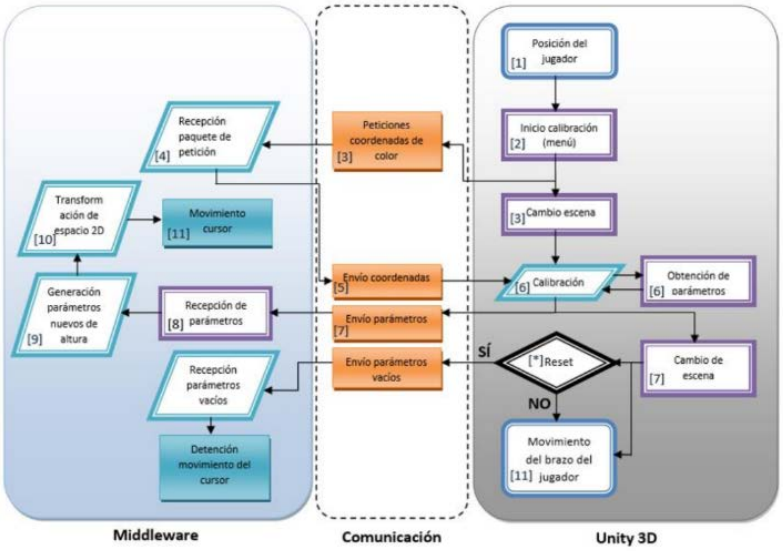


Figura 15. Diagrama de flujo para la solución del movimiento del brazo [17]

Para la realización de este proceso se han requerido una serie de cambios tanto en los *Scripts* del *Kinect Asset* para el juego como en algunos de los *Scripts* dentro del *middleware* K2UM. De esta manera, tras la integración entre el proyecto de Gerardo Cilleruelo [17] y el K2UM 2.0 implementado por Daniel Iglesias, se crea la última versión del *middleware* llamada K2UM 3.0.

Esta versión es la que se utiliza finalmente en *Phiby´s Adventures 3D*, ya que, aunque no se ha utilizado la solución para el movimiento del brazo para la detección del movimiento del cursor implementada, se tiene presente como una futura línea de trabajo, debido a que es de gran utilidad en áreas en las que se precise del cursor, como los diferentes menús o la pantalla de inicio de este.

2.3.3. Blexer-Med 2.0

Se actualizó la web terapéutica hacia su segunda versión Blexer-Med 2.0 por parte de Alba Aguilar [5]. Esta web está subdividida en tres tipos de permisos: Superadministrador, Administrador del centro y Terapeuta como aparece en la Figura 16. Por lo tanto, a la web se puede acceder de estas tres formas introduciendo el nombre de usuario y la contraseña.

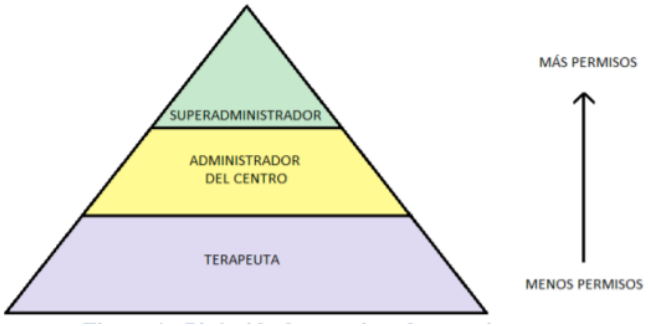


Figura 16. Pirámide de permisos de usuario [15]

Esta jerarquía tiene como propósito una mayor organización entre las personas implicadas en la interacción con los múltiples parámetros de la web. Para eso, los papeles que desempeñan las partes implicadas en esta son:

- Superadministrador (*Super Admin*): es el encargado de administrar todos los centros y, a su vez, cooperar con los administradores encargados de estos.
- Administrador del centro (*Center Admin*): tiene como propósito organizar a todos los terapeutas de su centro.
- Terapeuta (*Clinician*): su objetivo consiste en la colaboración con los pacientes implicados en la web para el establecimiento de los diferentes parámetros en cada ejercicio y la monitorización de los resultados.

En el caso del superadministrador (en la página aparece como *Super Admin)*, una vez el usuario ha introducido los datos pertinentes, la estructura de este se halla fraccionada en cuatro secciones: *Centers, Admins, Games* y *Exercises*. El superadministrador, al estar en la cumbre de la pirámide mostrada anteriormente, puede crear, editar y borrar cualquier Centro, Administrador, Juego y Ejercicio. Por otro lado, el administrador podrá tanto acceder como editar la información de su centro, además de añadir borrar y editar los terapeutas y los pacientes del centro en el que trabaja.

Por último, los terapeutas disponen de una plataforma donde pueden añadir, borrar y editar cualquier paciente con el que esté realizando la rehabilitación. De igual modo, cuentan con tres pestañas como las que aparecen en la Figura 17: *Patient, Exercises* y *Results*; estas pestañas son necesarias para el seguimiento en el tratamiento del paciente seleccionado.

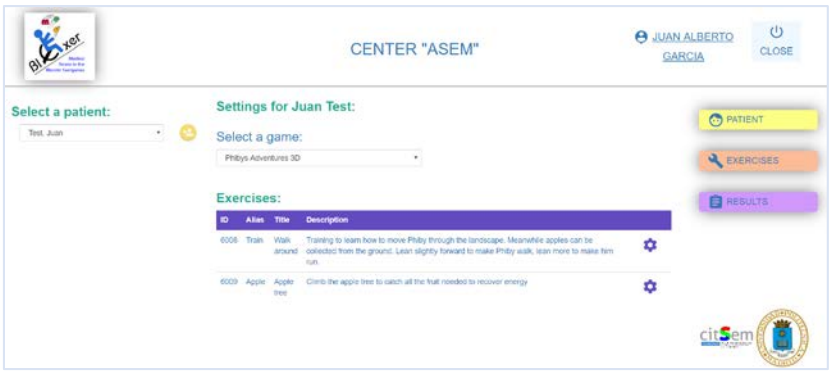


Figura 17. *Clinical Frontend* de la web

El papel del terapeuta es un enlace muy destacado en el desarrollo de este proyecto, puesto que su labor principal es la de crear un tratamiento de rehabilitación adaptado para cada paciente, en el cual interviene particularmente en cada uno de los ejercicios a realizar en el juego de *Phiby´s Adventures 3D*. La forma en la que se desarrolla este proceso, viene dada según los datos que este pueda manejar. Un ejemplo útil para la línea de este proyecto sería cuando el terapeuta, habiendo creado un paciente previamente, decide acceder al juego de *Phiby´s Adventures 3D*, donde escoge un ejercicio concreto. Después de leer la información sobre el ejercicio, el terapeuta rellena los diferentes campos que se utilizarán como variables para el juego. Además, puede guardar varias filas de parámetros y escoger en cada momento la que considere más adecuada para la realización de este

ejercicio. Posteriormente, tendrá la posibilidad de evaluar los resultados y modificar los parámetros de cada ejercicio teniendo en cuenta los progresos del paciente.

Este proyecto se realiza en contacto directo con la plataforma Blexer-Med *2.0*, debido a que se manejan los diferentes parámetros extraídos de la web por el *middleware*, para la composición y la alteración de las diferentes escenas elaboradas en *Phiby´s Adventures 3D*, las cuales se mencionarán más adelante.

2.3.4. *Phiby's exercises in* Unity 3D *(demo)*

El receptor de los datos enviados por el *middleware* consiste en un conjunto de componentes tipo *Script*, las cuales forman lo que se denomina el *Kinect Asset*, el cual se integra en una aplicación programada en Unity 3D, en este caso el juego *Phiby'sAdventures 3D*. Básicamente, los *Scripts* de este *asset* se encargan de recibir las coordenadas capturadas por la Kinect 2.0 y enviadas al K2UM, por lo tanto, sin el *asset* no sería posible valerse de este dispositivo para la utilización del juego. En la Figura 18 se representan los *Scripts* que componen el *Kinect Asset*. En el **Anexo I** es posible consultar las funcionalidades de algunas de las componentes, extraídas a partir de la documentación y comentarios creados por César Luaces [3] y actualizados por Daniel Iglesias.

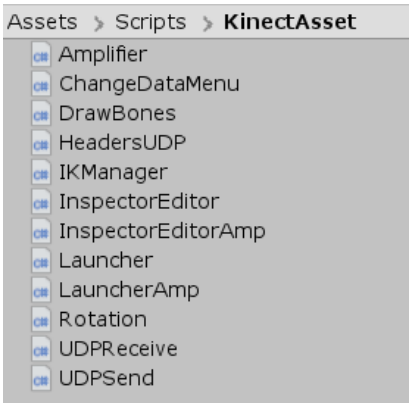


Figura 18. *Scripts* del *Kinect Asset* desde la aplicación

Durante el proyecto de creación del K2UM, se creó también una versión demo de *Phiby´s Adventure*, para demostrar el funcionamiento del *middleware*. Se compone de cuatro escenas de ejercicios en Unity 3D, basadas en la versión 1 de *Phiby´s Adventure*: *"Row the boat", "Chop the Wood", "Pick up the Balloons"* y *"Climb the tree"*. Como se puede ver en la Figura 19, se sustituyó la escena de *"Dive and eat"* por "*Pick up the Balloons"*, aunque en ambas se realizan los mismos tipos de movimientos mencionados en la sección de *Phiby´s Adventure v1*.

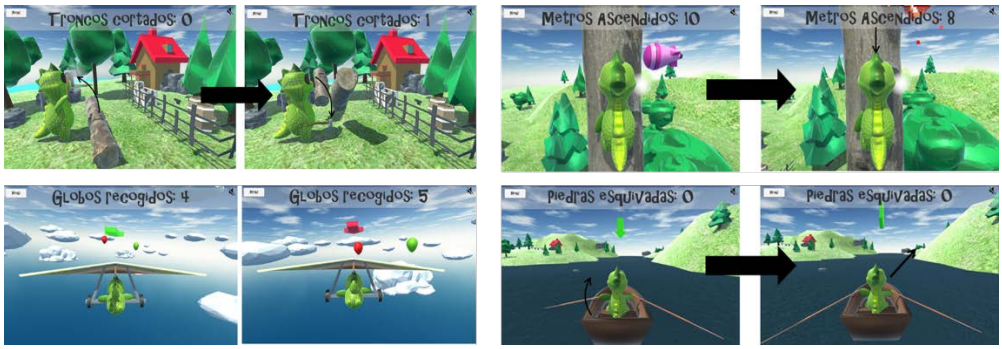


Figura 19. Escenas de *Phiby´s Adventure demo* de Unity 3D

2.3.5. *Phiby's Adventures 3D*

El objetivo principal de la versión del juego de *Phiby's Adventures 3D* es modificar la forma de jugar hacia un manejo más libre del personaje que en la primera versión, es decir, en vez de un movimiento restringido por celdas, se optó por un ambiente libre de un mundo abierto.

Se inicia el trabajo en esta versión en febrero del año 2019 con un grupo de trabajo de tres personas durante el semestre de primavera 2018-19:

1. Laura Molero - Diseño e implementación del entorno [18]

Inicialmente, se desarrolla el primer modelo de la isla del mundo abierto donde Phiby comienza su aventura. Para ello, a través de un *Heightmap* como el que se representa en la Figura 20 se crea el diseño inicial de esta y, posteriormente, gracias a las herramientas que Unity 3D dispone, se procede al ajuste de picos y la creación de colinas y montañas mediante el moldeado con el instrumento de brocha.

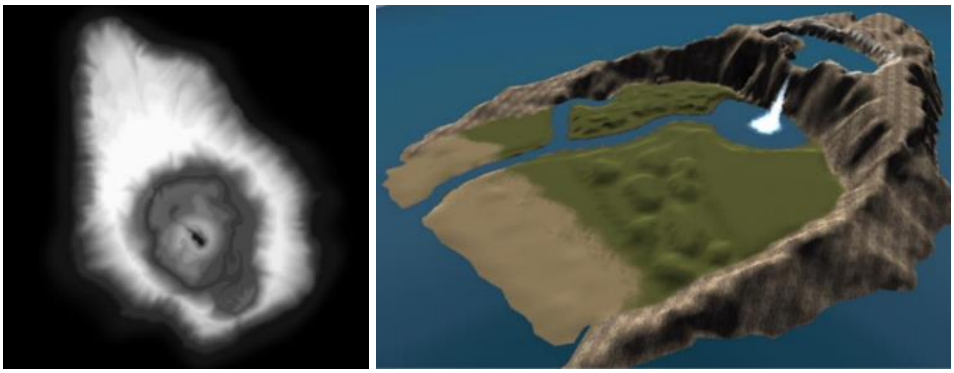


Figura 20. *Heightmap* y molde con texturas de la isla [18]

En segundo lugar, se crea el entorno donde habitan las plantas, los árboles y los animales que forman parte del mundo abierto, junto con una cascada, dos lagos y un río. De igual manera, se importa la cabaña de la versión anterior del juego, creada por Cristina Esteban Gómez [19] y, a través de un diseño del ambiente, se colocan estos objetos en las diferentes partes de la Isla, originando así un hábitat como el mostrado en la Figura 21.

Figura 21. Área de la zona de la cabaña de la isla

2. Miguel Ángel Gil - Integración y calibración de movimientos [20]

Posteriormente, se procede a la integración y calibración de movimientos. En primer lugar, se reducen las caras y vértices del personaje, asimismo, puesto que el tamaño del personaje era de 9 metros de altura, se disminuye el tamaño de este a 1 metro antes de la exportación (Figura 22).

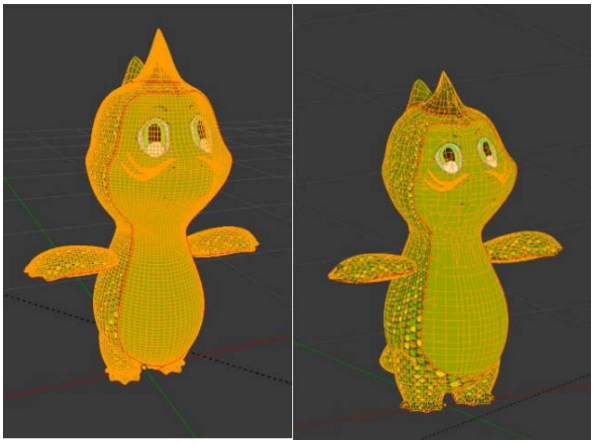


Figura 22. Muestra del antes y después de la malla del personaje tras la reducción de caras y vértices [20]

Tras el ajuste de la malla, en este proyecto se procedió a la creación de animaciones (Figura 23) en el mismo programa Blender. Seguidamente, se inició el proceso de exportación del personaje y se importó el fichero *.blend* en la nueva plataforma Unity 3D.

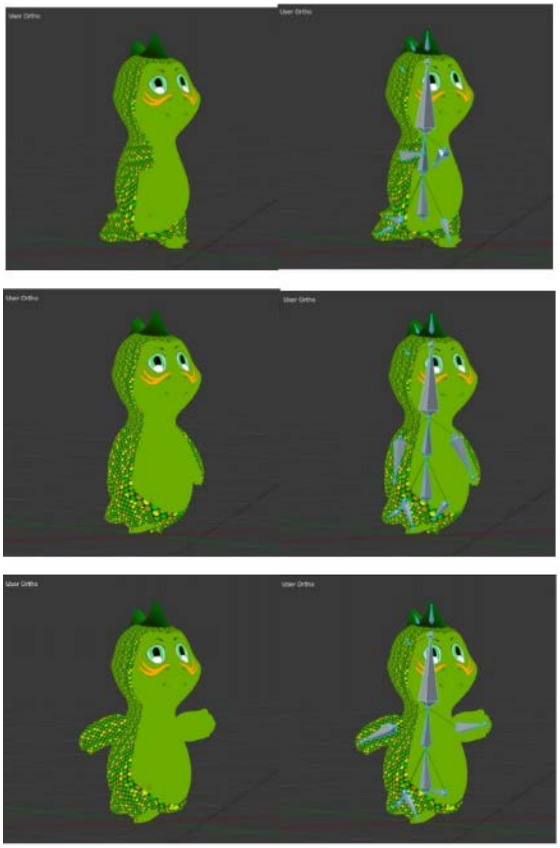


Figura 23. Ejecución de la animación en diferentes intervalos junto al esqueleto del modelo [20]

A continuación, y tras el periodo de pruebas de control del personaje realizado, el cual se mencionará en la solución desarrollada por este PFG, se implementó el primer *Script* de control del personaje mediante el teclado y, por medio de un gestor de animaciones de Unity llamado *Animator*, se consiguieron realizar las animaciones de "estado de reposo", "caminar" y "correr", como se muestra en la Figura 24.

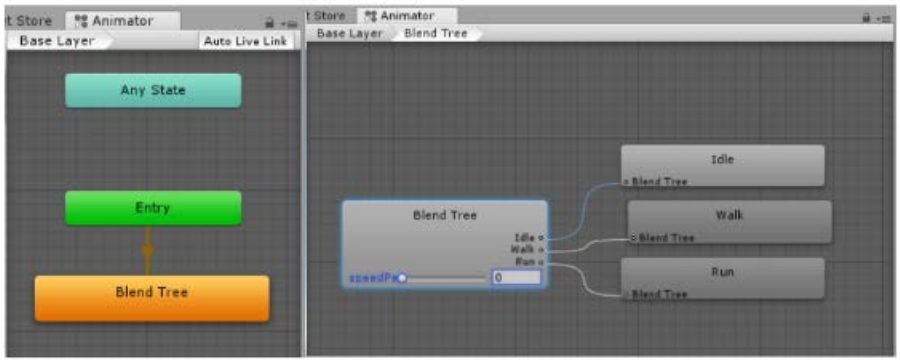


Figura 24. *Animator* asociado al personaje principal [20]

Por otro lado, puesto que no fue posible reutilizar las escenas de la demo, se crearon cuatro escenas a partir de lo que se consiguió sustraer de esta: "*Chop the Wood*", "*Pick up the Balloons*", "*Climb the tree*" y la nueva creación de "*Snow skiing*".

Para finalizar, con la colaboración de este PFG, la cual se mencionará posteriormente en la solución desarrollada, y gracias a la reutilización del *Script* de control del personaje aportado por Pablo López en su proyecto [21], se prosiguió con la asignación del

personaje Phiby al *Kinect Asset* de la primera versión del K2UM. Se puede acudir al estudio de la asignación del personaje al *Kinect Asset* gracias al documento redactado por Miguel Ángel Gil presentado en el **Anexo II**.

3. Juan Alberto García - autor de esta memoria

Fue el encargado de establecer una configuración a distancia, vía la web del terapeuta, de los cuatro ejercicios a integrar en el entorno. El desarrollo de los ejercicios fue tarea de Miguel Ángel, aunque no se ha podido solucionar en su totalidad, por lo que la integración de la web sufrió retrasos. Además, la actualización del *middleware* aún estaba por terminar, por lo que el autor de este trabajo se ocupó de contribuir con múltiples aportaciones de otro tipo, como:

   a. Creación de un "Mini mapa 2D" para la visualización de la posición del personaje
   b. Creación del prototipo del personaje Phiby con movimiento por la isla (colaboración con Miguel Ángel)
   c. Diseño de manzanas y reparto de ellos en el suelo para probar la recolección de objetos con el personaje
   d. Diseño e implementación de un Ala Delta como objeto para posible integración de un ejercicio de volar
   e. Diseño y elaboración de una cueva dentro del entorno de la isla
   f. Creación de textos de aviso al jugador, como realimentación con el encuentro con objetos activos

Estas aportaciones se describen en detalle en el apartado 2.3.6 "Aportaciones de diferentes elementos de utilidad al juego".

Después de esta fase inicial, la segunda fase de desarrollo del juego tuvo lugar durante el semestre de otoño del 2019-2020 con un equipo de trabajo diferente:

1. Juan Alberto García coge las riendas del desarrollo y, después de una fase de integración y limpieza profunda, asienta las bases arquitectónicas para el juego, descritas en detalle en el capítulo 3.
2. Haoru Qin (PFG) se ocupa de ampliar la historia del juego, añadir elementos nuevos para realizar mecánicas de juego como un contador de vida, un depósito de equipamiento para objetos coleccionables, nuevas partes del ambiente etc.
3. Yuxue Liang y Pengzhou Wu (en prácticas) aportan elementos nuevos para las escenas y el manejo de la escena de trepar el árbol.

2.3.6. Aportaciones de diferentes elementos de utilidad al juego

Mini mapa 2D

En este apartado se explica cómo ha sido el proceso de creación de un mini mapa, el cual muestra la información de la posición del jugador con respecto al mundo 3D que tiene a su alrededor disponible. La causa por la que se decide crear este objeto es sencilla, dado que es un instrumento muy ventajoso para la orientación del jugador en el mundo abierto. No obstante, antes se explica qué es un *Canvas* y cuál es su papel en este proyecto.

El *Canvas* es el espacio donde todos los elementos de la UI (*User Interface*) deben estar. Se trata de un *Game Object* 2D con un componente *Canvas* asociado. Además, todos los elementos UI deben ser hijos del *Canvas*. Asimismo, este objeto tiene un ajuste *Render Mode*, el cual puede servir para renderizar el espacio de la pantalla o el espacio del mundo 3D y un *Canvas Scaler* para la modificación del tipo de escala y la graduación que se necesite de esta. Véase en la Figura 25.

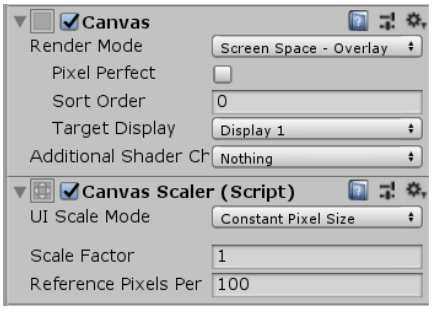


Figura 25. *Canvas* en el inspector

Cámara del mini mapa

Para la creación del mini mapa, primero se crea una cámara, la cual se sitúa justamente encima del jugador, con su eje frontal perpendicular al suelo, mirando al jugador desde arriba, como se muestra en la Figura 26.

Figura 26. Posición y rotación de la cámara del mini mapa con respecto al personaje

Por otra parte, como lo que se desea es que la cámara siga al jugador, esta se introduce como hijo del *Phiby Character*, a 10 unidades de altura (eje Y) de este y una rotación de 90 grados en x, véase la configuración necesaria en la Figura 27.

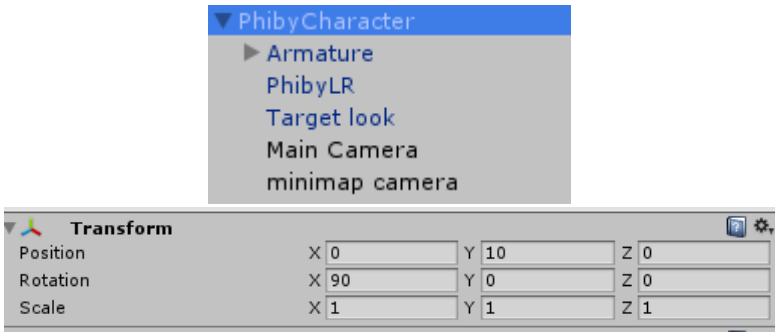


Figura 27. Configuración de la posición y la rotación de la cámara con respecto al personaje

Mini mapa en el *Canvas*

Un *Canvas*, por defecto, no puede contener una cámara, por lo que se utiliza una textura de renderizado en tiempo real. Esto es, un tipo de textura que, en lugar de venir pintada, se pinta a cada unidad de tiempo real.

Para hacer esto, primero se crea la textura de renderizado del mini mapa y posteriormente se arrastra al *Target Texture* de la cámara del mini mapa creada anteriormente, como se muestra en la Figura 28.

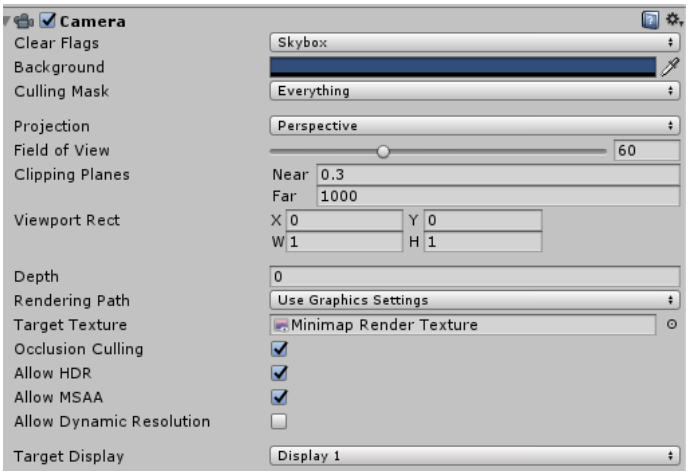


Figura 28. Configuración de la vinculación entre en *Minimap Render Texture* y la cámara del mini mapa

A partir de este momento se procede a la creación del *Canvas* (Figura 29), que es el elemento que contiene los objetos de la interfaz gráfica (textos, imágenes, botones etc.). Durante la creación, se genera automáticamente por defecto el *Event System*, el cual sirve para detectar la interactuación del Usuario con el *Canvas* (se sirve de este para el empleo del sistema de mensajes). Tanto el *Canvas* como el *Event System* forman parte de la UI.

Figura 29. Objetos de la UI

Por último, se crea una *Raw Image* (una imagen en formato crudo) en el *Canvas*, la cual sirve para mostrar la textura de renderizado del mini mapa creada anteriormente, se llama "*Minimap Raw Image*".

A la "*Minimap Raw Image*" se le asigna la textura de renderizado del minimapa, arrastrándola al campo *Texture*. De esta forma, aparece inmediatamente en el *Canvas*, dónde se puede escalar su tamaño y ubicar en cualquier posición en este (Figura 30).

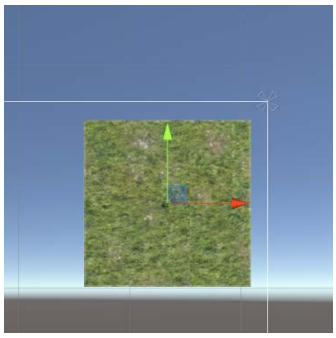


Figura 30. Posición del mini mapa en el *Canvas*

Unity posibilita modificaciones en la altura, el campo de visión de la cámara y el *clipping planes* (concepto que hace referencia a la distancia a la que comienza a grabar la cámara) para ajustar los detalles de forma personalizada para el programador.

Máscara y borde del mini mapa

La máscara tiene la función de dar forma al mini mapa, puesto que por defecto es un cuadrado. En este caso se hace empleo de un mini mapa circular.

Para la creación de la máscara circular del mini mapa, primero se crea una imagen de la UI dentro del *Canvas,* bajo el título *minimap mask*. Esta máscara requiere de una imagen, en este caso, se importa la imagen de un círculo negro con el fondo transparente. Después, se introduce la *minimap Image* como hija de *minimap mask*, y se posiciona la imagen encima de la máscara. En este caso es la posición (0,0,0), dado que es una relación padre-hijo.

Por último, se añade una componente *Mask* a la *Minimap Mask* para crear lo que se denomina "máscara de recorte", que genera la forma del mini mapa, como se puede ver en la Figura 31.

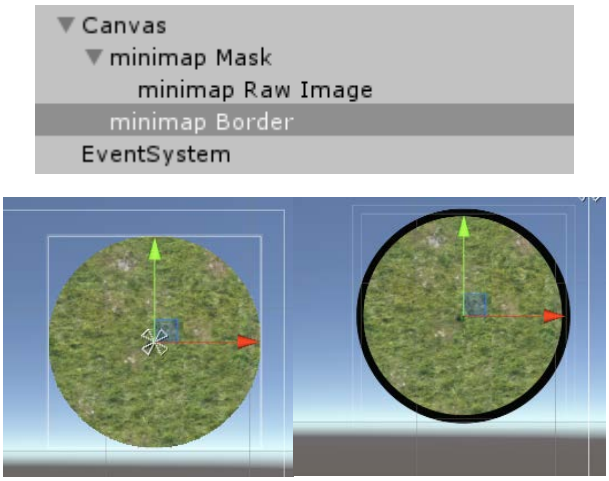


Figura 31. Aplicación de la máscara de recorte (izq.) y el borde (der.) del mini mapa

El borde del mini mapa es una forma muy sencilla de generar una estética de composición en la que se puede diferenciar mejor del mundo 3D. Para ello, se crea también una imagen dentro del *Canvas*, a la cual se titula *minimap Border*. Al igual que la máscara, hay que proporcionar un fichero de imagen. En este caso, se importa la imagen de una circunferencia negra con el fondo transparente y se coloca con el tamaño y posición exacta de tal forma que rodee a la *minimap Mask* en el entorno gráfico del *Canvas*.

Objetos 2D en el minimapa

Puesto que el objetivo del mini mapa en este proyecto consiste en mejorar la orientación del jugador en las diferentes zonas del mundo abierto, otra forma de optimización reside en la inclusión de los nombres de las regiones más importantes en el mini mapa, como son la cabaña, el río, la cueva, la cascada, etc. Además de un indicador verde que muestra hacía dónde está mirando el personaje en tiempo real, el cual se introduce dentro del personaje al que acompañe, en este caso, Phiby. Asimismo, se incluyen algunos de los objetos para indicar que Phiby puede interactuar con ellos (Figura 32).

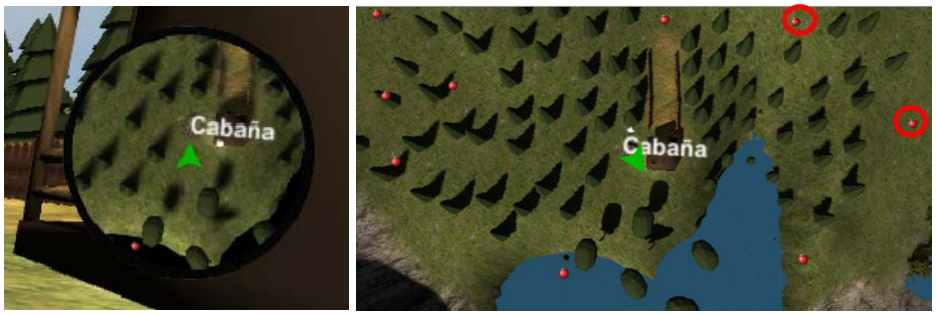


Figura 32. Visualización del mini mapa (izq.) y de los objetos añadidos en la escena (der.)

Para realizar esta parte, ha sido necesaria la obtención de imágenes vectorizadas, las cuales se introducen en el mundo abierto con orientación en el eje Y positivo del universo 3D (es decir, orientación vertical positiva). Al tratarse de objetos 2D, estos se posicionan entre cualquier objeto o porción del terreno 3D y la cámara del mini mapa; de esta forma, la cámara será capaz de captar estos objetos, debido a que no hay ningún otro objeto ocultando las imágenes (Figura 33)

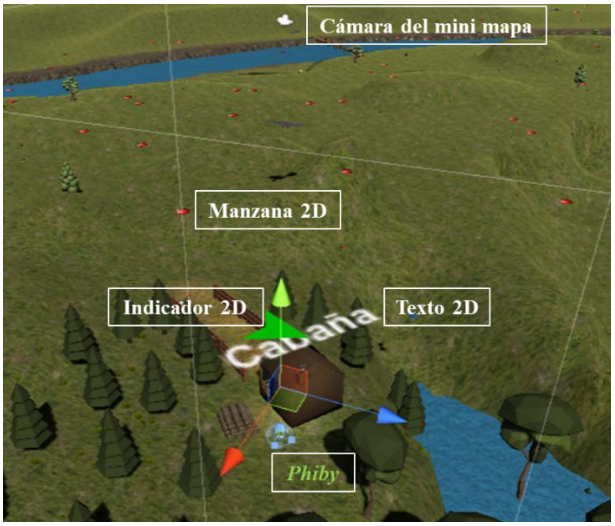


Figura 33. Visualización de las imágenes 2D posicionadas entre los objetos 3D y la cámara del mini mapa

Capa del mini mapa

Una vez se han introducido todas las imágenes 2D para la visualización de estas en el mini mapa, se procede a crear una capa del mini mapa (*Minimap Layer*). Esta capa será seleccionada en todos los objetos que se vayan a utilizar en el mini mapa. De esta forma, lo que se desea conseguir es que todo elemento que forme parte de esta capa, sólo se pueda percibir por la cámara del mini mapa. Véase en la Figura 34.

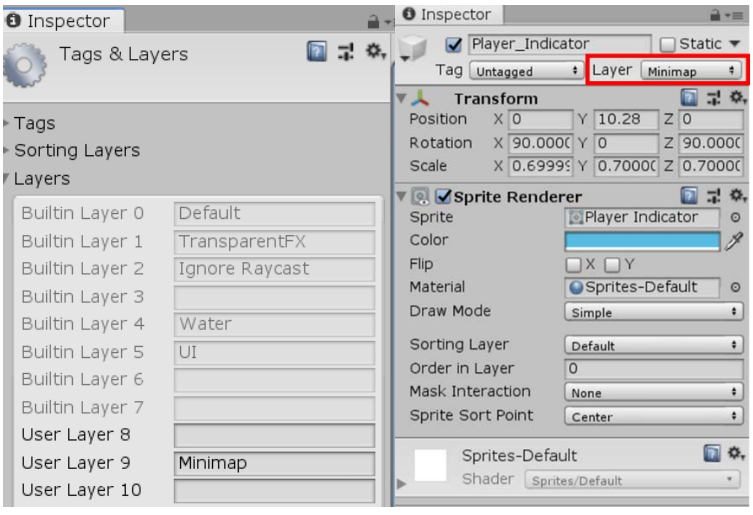


Figura 34. Creación del *Minimap Layer* y asignación de la capa al indicador de personaje

Por último, en la configuración de la cámara asociada al personaje principal, en este caso Phiby, se desmarca la capa del mini mapa, como se muestra en la Figura 35. De esta forma, no se capta ningún objeto que tenga esta capa asignada por la cámara del personaje.

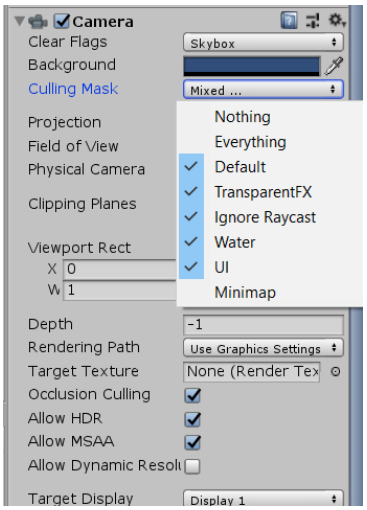


Figura 35. Asignación de capas visualizadas por la cámara del personaje principal

Prototipo del movimiento de Phiby

Para la creación del personaje principal del nuevo juego de *Phiby´s Adventure 3D*, previamente se recurre a los modelos establecidos en el proyecto de *Phiby´s Adventure*, en el cual se modela y texturiza al personaje diseñado por Cristina Esteban [19] en Blender.

Al comenzar la estructura de un nuevo juego en un mundo abierto, la primera tarea es exportar el Phiby 3D de Blender e importarlo a Unity 3D. De esta manera, para la importación se precisa de un gran control de esta nueva plataforma que se adquiere mediante diferentes pruebas.

Pruebas de importación y manejo de Unity 3D

Para ello, primero se comienza implementando un prototipo de movimiento de un personaje en primera persona sobre un terreno. Posteriormente se añade una cámara 3D de seguimiento del objeto y se importa (sin animaciones) el modelo de *Phiby 3D* exportado de Blender. De esta manera, y mediante *Scripts* de la *Asset Store* de Unity 3D, se controla a Phiby a través teclado y ratón.

Por otro lado, se implementó el movimiento del personaje principal descrito en el apartado 2.3.5 "*Phiby's Adventures 3D*". De la misma manera, se desarrolla la asignación del personaje principal a la Kinect 2.0 como parte de la integración en el proyecto de *Phiby´s Adventures 3D* (redacción del funcionamiento de este en el **Anexo II**) y teniendo como referencia el proyecto expuesto por Pablo López [21] en el cual se asigna el personaje principal *Fruitman* a la Kinect 2.0 mediante el *Kinect Asset* [3]. Para ello, se hace uso del *Script* implementado en el proyecto que utiliza un sistema de ángulos mínimos para el movimiento del personaje (Figura 36).

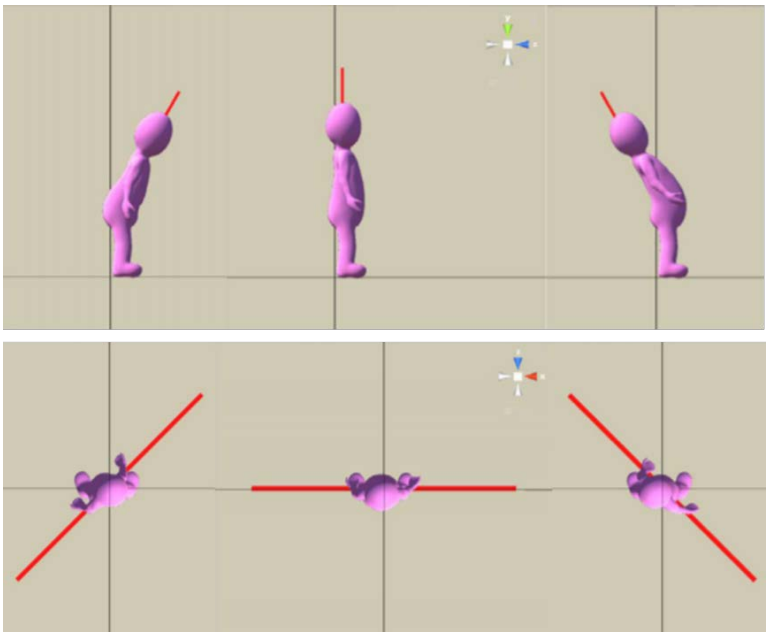


Figura 36. Sistema de ángulos mínimos para el movimiento del personaje [21]

Esto sólo es el inicio de un prototipo más amplio y estable, en el cual se trabajan otros aspectos de la animación del personaje para estado de reposo, caminar, correr y saltar representados anteriormente.

Manzanas en el suelo

El objetivo de esta parte del proyecto trata de comenzar a tener objetos en el terreno, los cuales sean útiles para recolectar y utilizar en posteriores misiones, o para ganar energía. De esta manera, se comienza utilizando un objeto de una manzana, la cual tiene su *Script* asociado para que Phiby sea capaz de recogerla y sumarla a la mochila de suministros. Para ello se utiliza un contador el cual se suma a una variable que deberá estar guardada en un *Script* indestructible tras los cambios de escena, puesto que, en caso contrario, se perdería el progreso y el contador se reiniciaría a cero en cada cambio de escena. Este *Script* se comentará en otro apartado de este proyecto (Diagrama de la Figura 37).

Otro propósito para este tipo de objetos es el uso de estos para el almacenamiento de energía del personaje principal una vez se haya fatigado tras realizar un ejercicio, durante un tiempo máximo dictaminado por los parámetros obtenidos de la web.

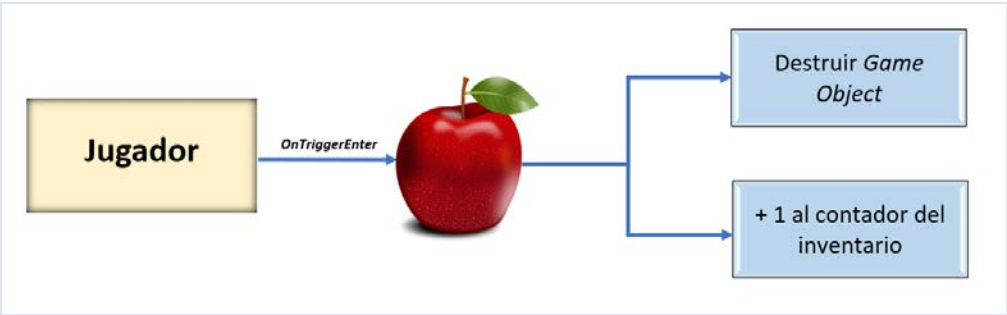


Figura 37. Diagrama de almacenamiento de objetos en el inventario

Ala Delta

Las dos formas principales de desplazarse para Phiby son caminado o corriendo. No obstante, como la isla es muy grande, se ha decidido buscar otras formas de viajar entre diferentes niveles de forma más rápida. Una de ellas es el Ala Delta, un objeto que ha servido para un ejercicio en la versión "Demo" de este juego y que ahora es útil no sólo para que el paciente pueda hacer un ejercicio concreto, sino también como medio de transporte.

Una vez obtenido el *prefab* del objeto, en este caso el Ala Delta del proyecto Unity 3D de César Luaces [3], se procede al desarrollo del *Script* asociado a este para simular el movimiento de un Ala Delta en el mundo abierto.

Scripts asociados

El *Script* llamado '**ControladorGlider.cs**' se implementa basándose en las características básicas de caída de un Ala Delta real, en el que se tiene en cuenta la velocidad de este en la dirección del plano XZ horizontal y los ángulos de rotación X (horizontal) Y (vertical) Z (sobre su propio eje). Figura 38.

Figura 38. Sistemas de avance y rotación del Ala Delta en el mundo abierto

Para la generación del prototipo del controlador se implementa en el *Update* del *Script* asociado al objeto el movimiento de avance sin lectura de teclado a una velocidad controlada por la variable *speed* modificable. Luego se genera la rotación horizontal mediante lectura de ratón X para la realización de giros de izquierda y derecha. Además, se genera la rotación del eje Z del Ala Delta (sobre su propio eje) por lectura de ratón X controlada por la variable *turnForward*. Este último paso se realiza para generar una sensación de giro real, puesto que un vehículo aéreo, cuando gira, rota sobre su propio eje de forma leve.

Después se procede a crear la rotación en el eje X (Vertical) automática, descendiendo de forma automática con una variable de control *fall*, que es la que rige el ángulo al que desciende por segundo. En este caso se le ha asignado un 0.05f a la variable.

Para finalizar el *Script* del controlador, si la lectura del ratón X es cero (el ratón no está en movimiento), se corrige la rotación del eje Z del Ala Delta mediante el método *Lerp* asociado a *Quaternion*, que realiza una interpolación entre la rotación actual y la rotación inicial (cero) para generar sensación de movimiento.

Otras componentes

Además del *Script* del controlador asociado al objeto, también se le asocia un *Capsule Collider* y un *RigidBody* con las características mostradas en la Figura 39 para el contacto del objeto con el resto del mundo abierto como un objeto que puede colisionar con el resto de los objetos y con el terreno de la isla.

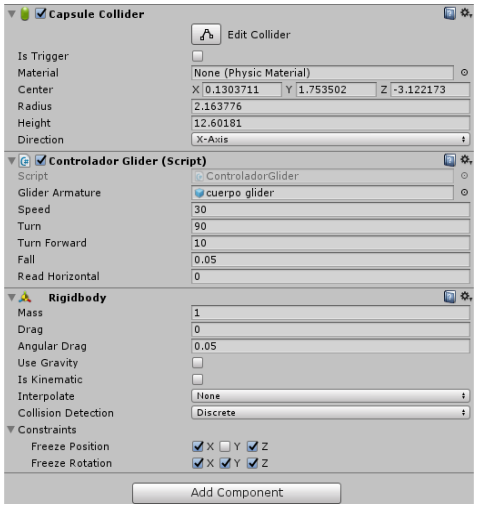


Figura 39. *Capsule Collider* y *RigidBody* del Ala Delta

Estructura del prefab

Como se puede ver en la Figura 40, la estructura del objeto *gmk_glider* se subdivide en:

- El cuerpo *glider*, dónde están todos los polígonos asociados a este.
- Dos sistemas de partículas de tipo *Afterburner* posicionados en la punta de las alas para proporcionar sensación de velocidad al usuario.
- Un objeto vacío llamado “Seguimiento”, el cual se explica más adelante.
- Otro objeto tipo *Camera* llamado *Minimap Camera*, el cual sirve para renderizar el minimapa que sigue al objeto desde arriba.

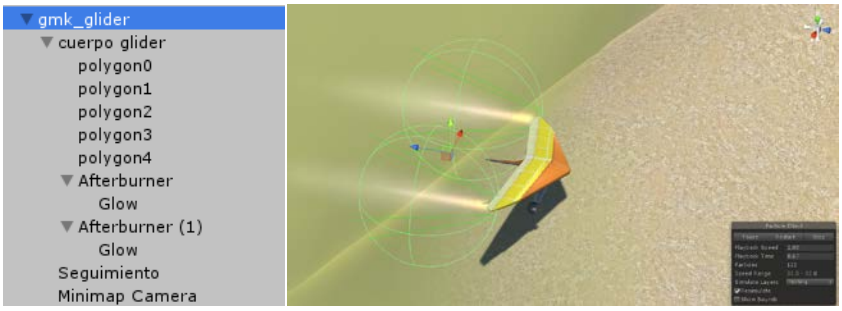


Figura 40. Estructura del objeto *gmk_glider*

Cámara

Se coloca una cámara en cualquier punto del espacio, la cual a través de un *Script* con el objeto vacío “Seguimiento” (Figura 41) visto anteriormente, se utiliza como entrada de este. A partir de este, de las funciones *transform* y el método *Lerp* para el *Vector3* (posición) y *Quaternion* (rotación), se produce el seguimiento de cámara mediante interpolación para generar una mayor sensación de movimiento, dado que lo

que realmente está sucediendo es que la cámara sigue al objeto Seguimiento con un pequeño retardo, tanto en posición como en rotación.

Figura 41. Cámara de seguimiento del ala delta

Asociación del Objeto al *Kinect Asset*

Después de haber entendido las físicas de este objeto, puesto que el Ala Delta está asociado a las coordenadas X del ratón, la asociación de este objeto al *Kinect Asset* se podría generar de dos formas:

- La primera, y seguramente la más compleja, sería activar el código proporcionado por el juego de *WestGun-Therapy* [17] para el movimiento del cursor en la pantalla controlado por Kinect. Este paso requiere el conocimiento previo de la asociación de una articulación activa al cursor y la calibración de este.
- La segunda solución es la de utilizar el *Kinect Receiver* proporcionado para el movimiento de Phiby en el mundo abierto con sus controladores asociados de forma exacta, de manera que se capte el eje horizontal corporal, sustituyendo de esta manera el control del Ala delta.

Cueva

Puesto que se trata de un juego de aventuras en un mundo abierto, la forma en la que se generan los límites del mapa es mediante el sistema de *respawn*. El *respawn* de un jugador consiste básicamente en la reaparición de este en un lugar concreto del juego cuando ha sobrepasado algún límite del juego o cuando ha perdido una vida y debe empezar desde un punto anterior. También puede ser motivo de *respawn* en los momentos en los que el jugador, por ejemplo, haya perdido toda su energía. Aunque las reglas de implementación del *respawn* están por definir, se trata de una futura línea de trabajo interesante para el proyecto, puesto que es una forma de conseguir un equilibrio entre la seguridad proporcionada al usuario de que está adquiriendo una serie de logros en el juego de aventuras, combinada con el riesgo de perder algunos de los avances.

Como se puede ver en la Figura 42 extraída del documento de *Yukai Chou's Octalysis Game-Techniques* [22], en este caso se intenta conseguir un equilibrio entre la parte de *Development & Accomplishment* en la que el jugador obtiene una serie de avances y

logros, con la sección de *Loss & Avoidance* en la que el jugador debe evitar que le ocurra algo malo (en este caso, que realice una acción que sea motivo de *respawn*).

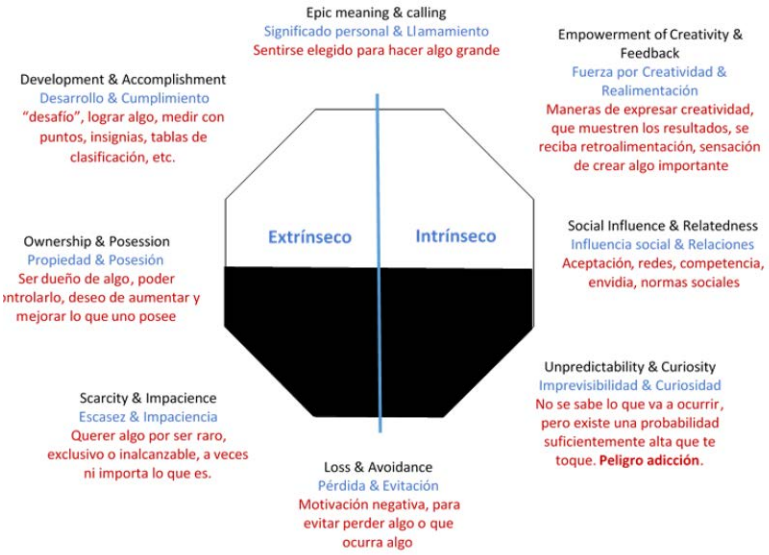


Figura 42. Diagrama extraído de *Yukai Chou's Octalysis Game-Techniques* [22]

De esta manera, se determina el diseño de un área de juego que es útil para el *respawn* del personaje principal cuando este haya caído en zonas que se consideren "Fuera de la zona de juego". Para ello, se requiere un lugar centrado en el terreno de juego, de manera que Phiby pueda volver de forma cómoda y rápida al lugar donde estaba. De este modo, se establece la Cueva como zona de *Respawn*.

Para el diseño de la Cueva se aprovechan dos partes elementales en Unity: la modificación del terreno y la incorporación de nuevos objetos.

Modificación del terreno

Para la modificación del terreno, se precisa de un estudio previo de este para así encontrar la localización adecuada para construir la cueva. Para ello, se tendrá en cuenta la forma y el tamaño de la zona de la isla para el primer nivel de *Phiby´s Adventure 3D*. En este caso, se ha elegido la zona rodeada con una circunferencia en la Figura 43, puesto que es un lugar más o menos céntrico en la isla y está dentro del área de juego para el primer nivel.

Figura 43. Localización de la cueva en el primer sector de la isla

Para la composición de la forma de la cueva, se acude a la herramienta de pintado de terreno que viene incorporada en la componente *terrain* de Unity (Figura 44). En esta herramienta vienen añadidos diferentes tipos de configuración para el trazado de altitud del terreno y el pintado de las diferentes texturas. Además, se pueden hallar diferentes tipos de brocha para la generación del relieve y el tamaño y la opacidad de esta es completamente configurable.

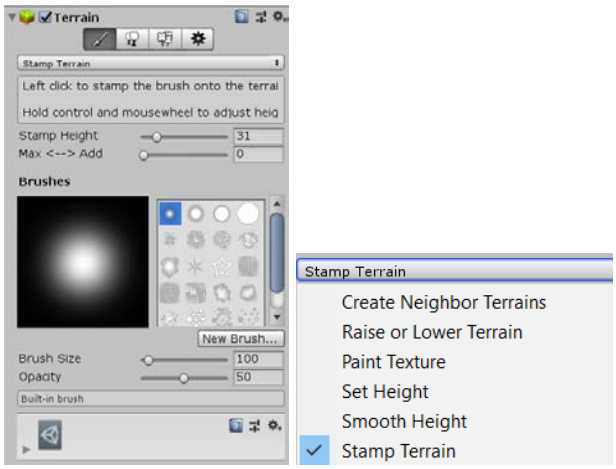


Figura 44. Herramienta de trazado de altitud y texturas sobre el terreno incorporada en Unity 3D

De esta forma, gracias a la versatilidad de la herramienta utilizada en Unity, se pueden amoldar las diferentes alturas del terreno con *Raise or Lower Terrain* y *Stamp Terrain* y a partir de ellas emplear una capa posterior de suavizado gracias al factor *Smooth Height*, obteniendo un resultado como el de la Figura 45. Posteriormente se le aplica una capa suave de texturizado de un material más rocoso que el que se utiliza en el resto de la cresta de la montaña, consiguiendo así un aspecto más parecido al de una caverna.

Figura 45. Resultado del terreno de la isla para la composición de la cueva

Incorporación de nuevos objetos

Se importan dos paquetes de objetos del *Asset Store* [23], uno de ellos es de rocas, las cuales se utilizan para generar la estética mejorada de las partes externa (Figura 46) e interna de las paredes y el techo de la Cueva y para oscurecer más el interior de la misma.

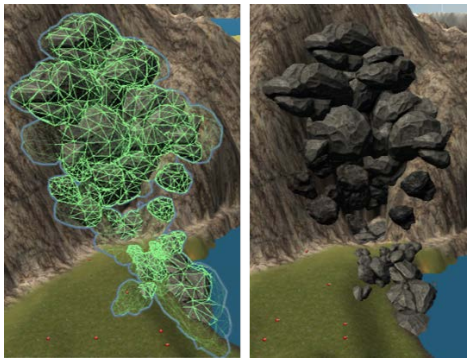


Figura 46. Aspecto externo de la cueva con la composición de las rocas añadida

Por otro lado, se importa una hoguera la cual genera luz de colores cálidos en el interior. Esta hoguera está formada por sistemas de partículas que vienen prediseñados en la *Asset Store* de Unity 3D, el cual reproduce virtualmente el movimiento del fuego. Un sistema de partículas es un tipo de animación en el que un número de partículas determinado interactúan entre sí y están sometidas a fuerzas externas. En la mayoría de los casos, se hace uso de este tipo de sistemas para la generación de fluidos, telas, combustión, etc.

Se utilizan seis sistemas de partículas para generar el efecto de la llama de la hoguera, los cuales están numerados en la Figura 47. Estos están mezclados para generar más realismo en el momento en el que se transita alrededor de ella.

- El primero sirve para la creación de la llama que produce la hoguera.
- El segundo genera una animación de brillo.
- El tercero y el cuarto son generadores de humo negro y blanco respectivamente.
- El quinto y sexto son animaciones de chispas descendiendo y ascendiendo respectivamente.

Figura 47. Objetos de generación de partículas para la composición de la hoguera importada de la *Asset Store* de Unity 3D

Textos de aviso al jugador

Por último, se implementa la sección que va ligada a los avisos de texto al jugador durante la partida. Estos avisos son una característica indispensable para mantener el interés del usuario y así evitar que este se desubique en el mapa a causa de su gran tamaño. Para ello, en la entrada de las escenas, las cuales están asociadas a un objeto concreto, se añade un *empty Game Object* como hijo del objeto en el cual, a través de un *collider* se le proporcionan instrucciones al jugador por pantalla, con ayuda del *Canvas*, de cuál es su próximo objetivo. En el ejemplo de la Figura 48, se utiliza como objeto principal un árbol donde el paciente puede trepar y conseguir manzanas.

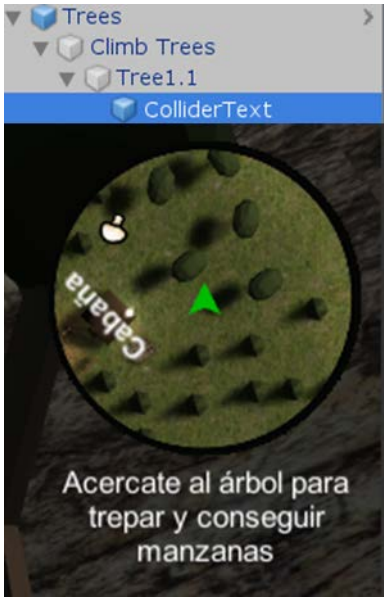


Figura 48. Ejemplo de un texto de instrucciones para el jugador

En la Figura 49 se puede ver un árbol con el *collider* asociado al objeto hijo de este, llamado *ColliderText*. A su vez, este objeto tiene asociada una componente *Script*, que es

la encargada de generar el comportamiento del objeto para indicar el mensaje que se quiera transmitir en el *Canvas.* Una vez el personaje principal traspasa este *collider*, se realiza la acción y, por tanto, se muestra el mensaje por pantalla. El prototipo de texto que se diseñó en esta parte, se aplica en múltiples sitios del juego.

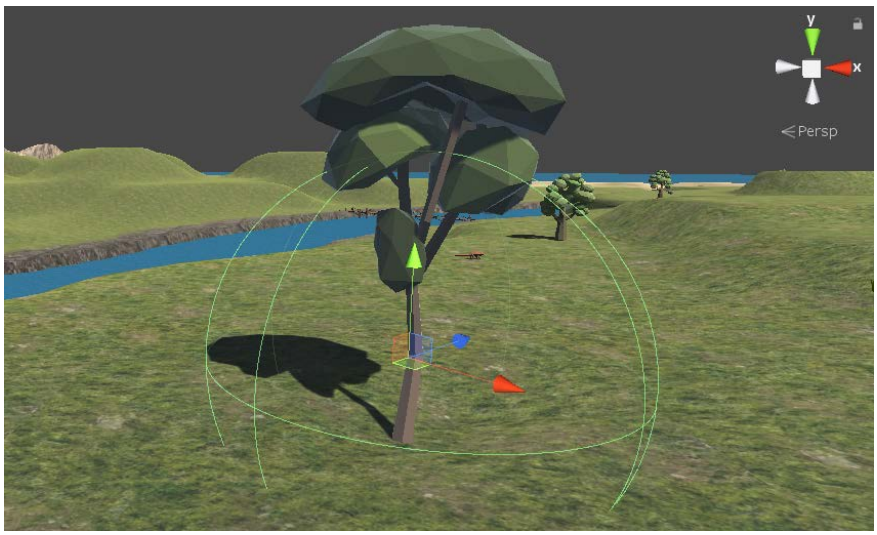


Figura 49. Objeto con *collider* asociado al árbol

3. Solución desarrollada

3.1. Integración de parámetros entre proyectos

Durante la elaboración de la arquitectura general del proyecto, se requiere de un continuo proceso de integración de parámetros entre proyectos, de cara a generar una coherencia global en la creación del juego. De este modo, en cada proceso de integración se han tenido en cuenta:

- La utilidad de los componentes a utilizar, por lo que ha sido necesario un proceso de limpieza, borrando así las partes innecesarias para este juego y manteniendo únicamente lo que es o va a ser de utilidad en un futuro cercano.
- El proceso de implementación de los *Scripts*, para de esta manera actualizarlos y adaptarlos a este proyecto.

Para el correcto desarrollo de algunas de las partes fundamentales de este PFG, se ha requerido de una progresión en el desarrollo del *Phiby´s Adventure 3D*; para ello, se recurre a proyectos finalizados con anterioridad que, tras su anexión a este y por medio del proceso de integración descrito, han sido beneficiosos para la creación de una versión estable a nivel práctico.

La consolidación del proyecto se genera a partir del proyecto creado por Laura Molero [18], donde se procede a la importación del resto de avances creados. En este caso, se decide integrar aquí, dado que es más sencillo integrar objetos creados a partir de programas externos con componentes añadidas y *prefabs* en una isla prediseñada en el propio motor de Unity 3D.

Este proyecto es la base principal del juego, donde a partir del inicio de la integración, se sientan las bases del juego titulado *Phiby´s Adventures 3D*, proporcionadas en el apartado anterior.

En segundo lugar, se decide añadir la nueva versión del personaje Phiby, tanto en la versión mediante control de Kinect como en la versión de control por teclado terminada por Miguel Ángel Gil. Esta versión también está constituida por la composición de las diferentes animaciones aportadas por el mismo autor en Blender.

Para la creación del personaje mediante el control de Kinect se añade el *Script* creado por Pablo López en su proyecto [21] y se aplica la solución proporcionada por el autor a la que se refiere en el apartado 2.3.6 sobre el prototipo del movimiento de Phiby. Posteriormente, en el proyecto desarrollado por Miguel Ángel Gil, se actualiza este *Script* y se implementan los cambios pertinentes para la mejora de la experiencia del usuario en el manejo de las mecánicas de juego del personaje principal.

Desde otra perspectiva, se inicia el proceso de importación de las escenas actualizadas en el proyecto de Miguel Ángel Gil, a las cuales se hace referencia en la sección de “Antecedentes” y servirán como prototipo para los diferentes cambios de escena implementados durante el transcurso de este proyecto, los cuales se mencionarán posteriormente en detalle.

Posteriormente, se inicia el proceso de integración de las componentes del *Kinect Asset* para la comunicación con el K2UM 2.0 con la nueva versión desarrollada por Daniel

Iglesias. Este procedimiento de anexión al juego de *Phiby´s Adventure 3D* es de gran complejidad, a causa de que se dispone múltiples parámetros que se necesita comprender y asociar para el correcto funcionamiento del juego. Puesto que durante la ampliación de *WestGun-Therapy* se ha requerido de una expansión de los *Scripts* del *Kinect Asset*, además de que a su vez estos *Scripts* en un inicio estaban ideados para este juego (*WestGun-Therapy*), se ha visto preciso mantener esta misma versión, a pesar de que las secciones de código de *WestGun-Therapy* no se utilicen actualmente. Se ha decidido proceder de esta forma, dado que estas funcionalidades servirán para futuras líneas del proyecto de *Phiby´s Adventure 3D* y, además, el propósito es crear un mismo *Kinect Asset* que sirva para cualquier videojuego.

Conjuntamente, la utilidad de algunas componentes como el **exerciseManager.cs** ha sido la de la exportación exclusiva de los parámetros aportados por la web a través del **settingSustain.cs**. De esta forma, invocando al **exerciseManager.cs,** y a través del *parseo* de las variables recibidas, es posible el aprovechamiento de estas para su posterior empleo en funciones relacionadas con el juego. Además, todo este proceso se ha visto beneficiado por el documento del Resumen del funcionamiento de K2UM llevado a cabo por Daniel Iglesias (**Anexo III**), el cual explica de forma detallada el funcionamiento de la última del nuevo *middleware*.

Tras la finalización del método de integración del nuevo *Kinect Asset*, se descubrió que durante la actualización desarrollada, gracias a la depuración de código llevada a cabo por Daniel Iglesias, y tras probar estas componentes en el proyecto de César Luaces [3], las escenas desarrolladas de esta demo, las cuales no se consiguieron aprovechar hasta la fecha, serían de gran utilidad en *Phiby´s Adventure 3D*, por lo que se inició el proceso de importación de las escenas de la demo configurada por César Luaces [3], ya que se consideró que el funcionamiento de estas era más óptimo que el de las aportadas en proyectos posteriores.

Tras la resolución de cada proceso de integración se ha tenido en cuenta la utilidad de cada componente importada mediante un proceso de limpieza metódico y reiterado, requerido para la progresión del proyecto en manos de otras personas que tengan como propósito el establecimiento de nuevas vías de trabajo en el juego.

3.2. Diseño de la arquitectura del juego

En este apartado se explican las diferentes técnicas para el diseño de una arquitectura de juego estable y compatible con las diferentes formas de trabajo de los programadores y diseñadores del juego de *Phiby´s Adventure 3D*.

Aunque no es el objetivo principal de este proyecto, el diseño de esta arquitectura es necesaria para generar una jerarquía ajustada al juego, e integrar los parámetros necesarios para este en base a los diferentes proyectos realizados anteriormente.

3.2.1. Diagrama de bloques general del proyecto

En la Figura 50 se muestra un diagrama de bloques resumido, culmen de la integración de los diferentes proyectos realizados con anterioridad ligados a la solución proporcionada por este PFG. Este diagrama sirve para entender lo que se explicará en los capítulos posteriores y, asimismo, proporciona una visión general del proyecto *Blexer* en el que se muestran de forma clara las cuatro partes en las que se divide:

- La web Blexer-Med 2.0 en color rojo.
- La Microsoft Kinect 2.0 en morado.
- El *middleware* K2UM *3.0* en naranja.
- El videojuego *Phiby´s Adventure 3D* en amarillo. Dentro de este, están:
  - Las escenas en verde: la isla y los ejercicios.
  - El *Kinect Asset* en gris.

El objetivo principal de este PFG se centra en gran parte en esta última sección, en la que participan las siguientes piezas del diagrama, algunas de las cuales ya se han explicado y otras que se mencionarán con mayor detenimiento en capítulos posteriores y que, entre otras, se consideran las más importantes:

Kinect Asset:

Tras la integración del nuevo *Kinect Asset* al juego de *Phiby´s Adventure 3D*, para su uso con el *middleware* K2UM 3.0, se consigue una completa comunicación con la web que servirá para el progreso del juego. Además, gracias al método *finishAndDestroy()* proporcionado por la componente **rotation.cs**, es posible el uso de la Kinect 2.0 tras el cambio entre escenas, del cual se explicará su aprovechamiento en el capítulo sobre el cambio entre escenas.

Exercise Manager:

Recogida de parámetros de configuración para el juego (de la web o de la plantilla).

Game Manager:

Instancia de los objetos *Game Parameters Template* y *Game Setting Sustain* y agregación de sus respectivas componentes antes del inicio del juego (*Awake()*): **gameParametersTemplate.cs** y **gameSettingSustain.cs**. Creación de variables para el juego a partir de los parámetros obtenidos y guardado en **gameSettingSustain.cs**.

Game Parameters Template:

Contiene la plantilla de parámetros por defecto, en caso de que no haya conexión con la web. Se trata de un objeto no destructible.

Game Setting Sustain:

Objeto no destructible que contiene los parámetros que se deseen conservar durante la partida. A este se puede acceder desde todas las escenas.

Island:

Escena del mundo abierto de la isla en la que se encuentran, entre otros muchos, los objetos dinámicos necesarios para el cambio de escena. Desde ella se puede acceder a otras escenas como "*Climb the Apple Tree"* y "*Chop the Wood"* siempre que se cumplan ciertas condiciones de apertura, las cuales se explicarán de forma más extensa en apartados posteriores.

Climb the Apple Tree:

Esta es la primera escena en la que se diseña una lógica de ajuste terapéutico, a partir de la fijación de los parámetros de la web.

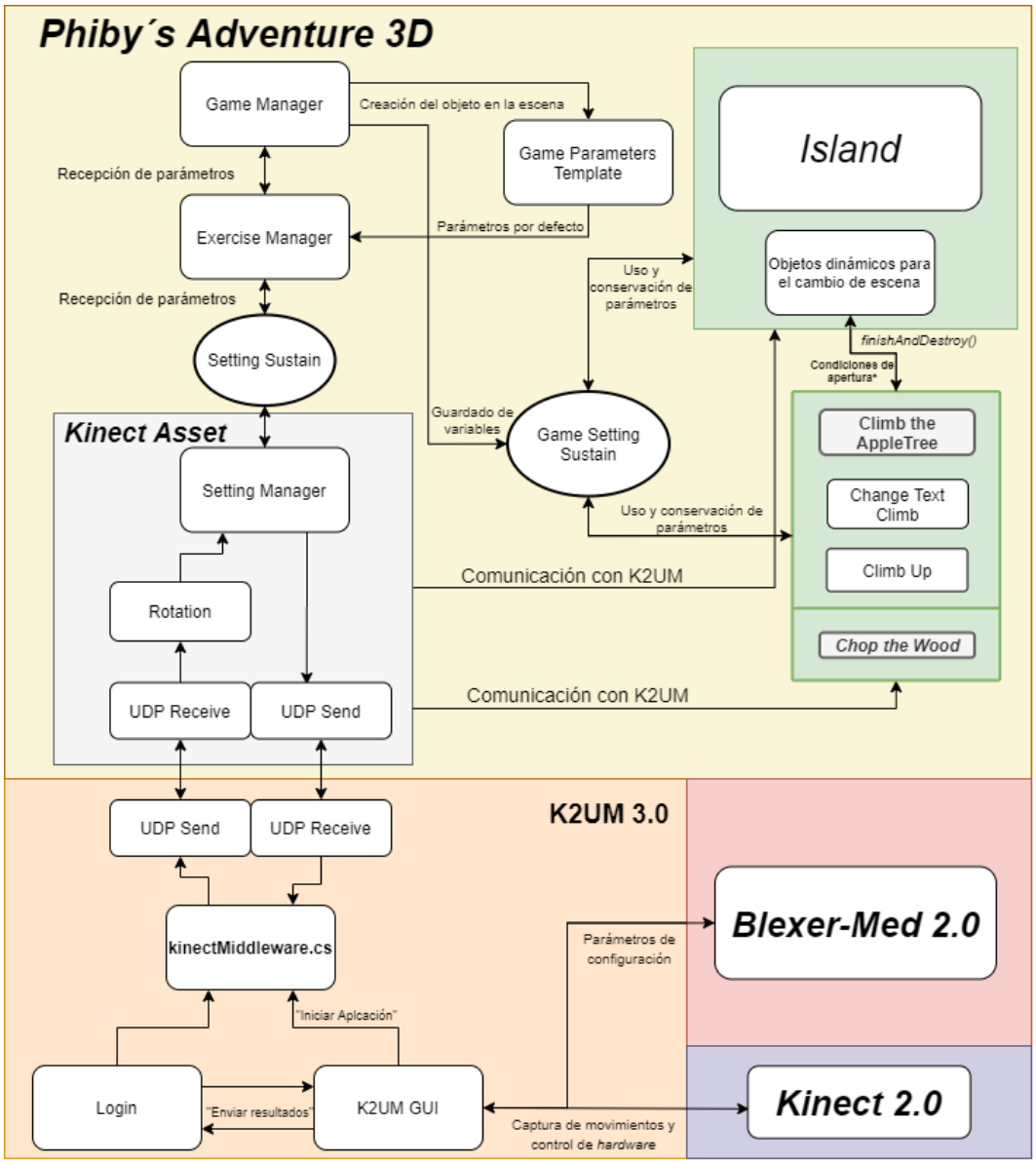


Figura 50. Diagrama de bloques general del proyecto

3.2.2. Estructura de proyecto genérica y manejable

Resulta imprescindible hacer hincapié en la estructura de proyecto, dado que en la creación del proyecto *Phiby´s Adventures 3D* han participado y participarán de forma paralela un número elevado de personas. La atención a la estructura y jerarquía de las diferentes áreas de juego es clave para conseguir un entorno de trabajo más manejable.

Las dos áreas principales a operar para conseguir esto en Unity 3D son las de *Hierarchy* y *Proyect*, las cuales se muestran en la siguiente Figura 51.

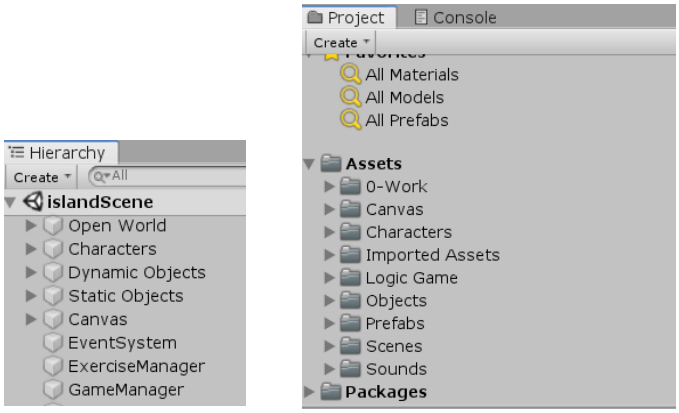


Figura 51. Jerarquía y organización del proyecto de *Phiby´s Adventure 3D*

Hierarchy

En esta área están los objetos utilizados en una escena, visualizados en forma de árbol, los cuales se organizan de forma jerárquica. En el caso de la Figura anterior, se muestra un ejemplo de la escena *islandScene*, que es el mundo abierto dónde el jugador vive su aventura y donde ocurren la mayoría de los acontecimientos del juego. Se ha escogido este *Hierarchy* dado que, al ser esta la escena principal, dispone de más contenido y, además, el resto de escenas se organizan de forma muy semejante a esta.

El modelo de jerarquía principal requerido en este proyecto para el manejo sencillo de las diferentes partes de la isla, se ha subdividido de la siguiente forma:

- *Open World:*
  - *Island Terrain:* Terreno total de la isla.
  - *Directional Light:* Luz utilizada para iluminar el mundo abierto completo.
- *Characters:* Por otra parte, se genera una carpeta donde están los personajes utilizados en la escena. Estos siempre están divididos en:
  - *NPC*: Personajes no jugables, algunos serán utilizados para comunicarse con el personaje principal, otros para dar pistas de por dónde continuar en la historia del juego y otros simplemente estarán como parte del mundo abierto.
  - *Main Character:* Personaje principal controlado por el usuario, en este caso se puede utilizar mediante teclado o Kinect según se estime. La idea del proyecto es que a través de una tecla se pueda conmutar entre control por teclado y por Kinect.

- *Dynamic Objects:* Estos objetos con frecuencia tienen un *Script* de comportamiento asociado, por lo que son dinámicos. En consecuencia, sus características pueden variar según la interacción entre el personaje principal y con estos. Algunos incluso pueden llegar a ser destruidos o aparecer de forma espontánea. En algunos casos pueden ser llamados y modificados por otros *Scripts*.
- *Static Objects:* Estos objetos son ajenos al personaje principal, por lo que sólo tendrán asociado un *collider* y formarán parte del entorno
- *Canvas:* en el interior de esta carpeta entrarán todos los objetos 2D manejados en el *Canvas*.

Fuera de esta jerarquía, por un lado, se encuentran los objetos *ExerciseManager* (creado por Daniel Iglesias), *GameManager* y *EventSystem*, los cuales tienen un papel clave en el funcionamiento global y se explican en detalle más adelante.

Proyect

Por otro lado, se encuentra la sección del proyecto en la cual se almacenan todos los datos utilizados en el juego de *Phiby´s Adventure 3D*. Aunque en el comienzo no se le da tanta importancia, puesto que hay muy pocos archivos que ordenar, en el transcurso del periodo inicial, en el que se forma el primer equipo de trabajo, se considera esencial la creación de una estructura del proyecto estable e intuitiva. Para ello, se requiere de un análisis completo de este a fin de que, de esta manera, una vez se hayan comprendido las funciones requeridas por el proyecto, se proceda a una intensiva limpieza y reorganización de todas las partes involucradas, incluidas las del proceso de integración.

Como se puede ver en la Figura 52, el proyecto se crea con la siguiente estructura:

- ***0-Work:*** Se crea esta carpeta antes del comienzo del segundo equipo de trabajo con el propósito de crear las bases para lograr un procedimiento de integración más eficaz. En esta carpeta se encuentran las sub-carpetas de trabajo de los alumnos que participan actualmente en el diseño. El equipo debe trabajar solamente en estas carpetas, es decir, crear o importar nuevos objetos e implementar su código aquí. En cada reunión semanal, se prueban los elementos nuevos (modelos, textura, *Scripts*, sonidos, etc.) y se mueven los que ya son definitivos a las carpetas adecuadas del proyecto general, manteniendo la estructura que se expone.

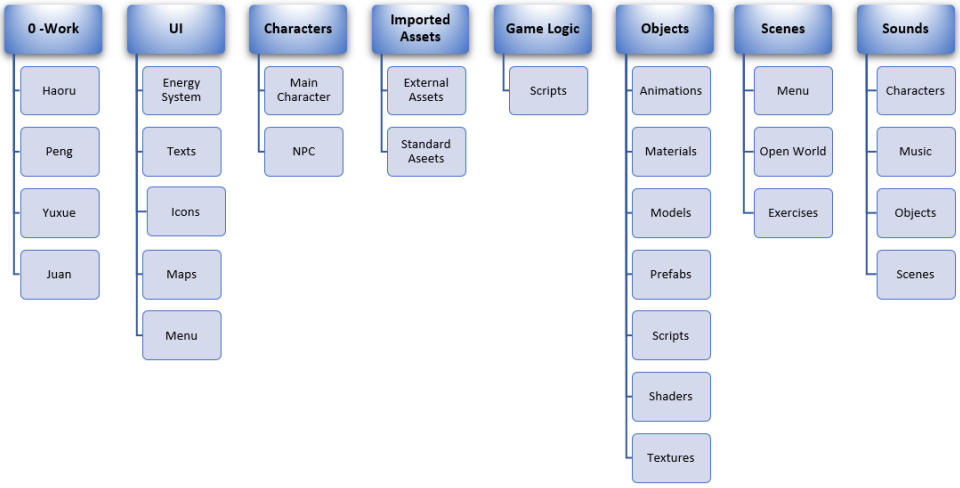


Figura 52. Estructura de los directorios del proyecto *Phiby´s Adventure 3D*

Reglas de trabajo a cumplir por el equipo:

- Nunca crear nuevos elementos fuera de la propia carpeta de trabajo.
- No mover elementos ya existentes a la carpeta de trabajo.
- Utilizar elementos ya existentes si se necesitan (p.ej. texturas…) sin hacer una nueva copia.
- Si es necesario editar un objeto ya existente, apuntar en un bloc de notas las modificaciones realizadas en dicho objeto.
- Añadir a cada *Script* una cabecera estándar (Figura 53) que recoge el nombre del autor, fecha de creación, el propósito del *Script* y espacio para modificaciones posteriores.

```
//------------------------------------------------------------------------
// PhibyKeyboardController
// Purpose: This script contains the logic to control Phiby with the keyboard
// Author:
// Date:
// ------------------------------------------------------------------------
// Modifications:
// Date - author - description
//
// ------------------------------------------------------------------------
```

Figura 53. Cabecera ejemplo para la organización de *Scripts* [24]

Las bases de integración han sido acordadas por Juan Alberto García y Martina Eckert y se recogen en el documento de diseño de juego o *Game Design Document* (*GDD*) en el que los miembros del equipo contribuyen a su redacción.

- ***UI:*** En esta carpeta se encuentran todos los ficheros incluidos en el *Canvas* para todas las fases del juego. En esta carpeta se incluyen cinco subcarpetas que almacenan todo lo relativo a los sistemas de energía (barra que indica cuánta energía le queda al personaje), textos emergentes (mencionados en el apartado 2.3.6), iconos utilizados tanto en los mini juegos como en el mundo abierto, mapas (incluyendo el conjunto de ficheros utilizados para la creación

del mini mapa) y ficheros empleados para la creación de los diferentes menús del juego.

- ***Characters**:* esta carpeta contiene todo lo necesario para la importación de los personajes *NPC* y el personaje principal, con *Prefabs, Models, Components (Animators, Scripts…),* y todos los elementos utilizados para la composición del *Kinect Asset* incluídos.
- ***Imported Assets***: En este directorio están los *Assets* importados de la *Asset Store* de Unity 3D y externos.
- ***Game Logic**:* Esta carpeta contiene las diferentes componentes utilizadas para la creación de una lógica de juego, desde el control de cámaras hasta el gestor del juego completo.
- ***Objects**:* Componentes (*Scripts, Animators, Animations…*), materiales, modelos, *Prefabs*, *Shaders* y Texturas útiles para la creación de objetos de las diferentes escenas.
- ***Scenes**:* Donde se encuentran las escenas para los diferentes menús, de la isla y de los minijuegos a los que accede el personaje en el transcurso de la partida.
- ***Sounds**:* En esta carpeta están todos los ficheros utilizados para la ambientación sonora del juego y la creación de un ambiente de inmersión.

3.2.3. Arquitectura del juego y sus principales componentes de control

La estructura global del juego en Unity 3D se divide en nueve partes: Entorno, Escenas, Objetos, *Canvas*, Personajes, *Game Manager, Exercise Manager*, Componentes no destructibles y Comunicación (*Kinect Asse*t con Blexer-Med).

Entorno

Isla (*Island Terrain*): se trata de un objeto con la componente *terrain*, modificado mediante las herramientas de entorno que vienen implementadas con Unity 3D.

Puesto que los objetos del entorno se crean directamente desde la jerarquía (*Hierarchy*) en la escena, estos no aparecen en los directorios del proyecto mencionados en el apartado anterior (*Proyect*).

Escenas

- *Climb the Apple Tree*: Entorno separado que contiene un árbol de manzanas y el personaje principal que debe subirlo mediante el movimiento alterno de los brazos impuesto por el jugador.
- *Chop the Wood*: Entorno separado que contiene la cabaña y el personaje Phiby delante de un montón de troncos. Aquí, el jugador debe conseguir que Phiby corte los troncos mediante un movimiento de un brazo.
- *Snow skiing*: Entorno separado que contiene una colina y el personaje Phiby con esquís puestos. El ejercicio a llevar acabo aquí por el jugador consiste en bajar esquiando la colina mediante la rotación lateral del tronco.

En los directorios del proyecto mencionados en el apartado anterior (*Proyect*), todo lo referente a las escenas está situado en la carpeta *Scenes* de forma ordenada.

Canvas

El motor divide la pantalla de juego en dos capas fundamentales, una es la que muestra la cámara utilizada en el mundo 3D y, por otro lado, está el *Canvas*, una capa 2D que tiene varias funcionalidades. En el *Canvas* se implementan el mini mapa presentado en el apartado 2.3.6, y en un futuro el inventario, la barra de energía, el menú y muchas otras características que se deseen añadir. En los directorios del proyecto mencionados en el apartado anterior (*Proyect*), todo lo referente al *Canvas* estará situado en la carpeta UI de forma ordenada.

Objetos

Los objetos pueden ser:

- Estáticos: se trata de objetos sin un comportamiento concreto, por lo que no llevan una componente *Script* asociada a ellos, estos suelen tener un *collider* y en algunas ocasiones un componente de animación.
- Dinámicos: son aquellos objetos que tienen uno o varios comportamientos en la isla y/o con respecto al personaje principal, estos casi siempre tienen una componente *Script* asociada la cual marca el comportamiento del objeto. En los objetos dinámicos, se pueden producir cambios de escena, llamadas a otros objetos y creación/destrucción de otros objetos, incluso del propio objeto.

En los directorios del proyecto mencionados en el apartado anterior (*Proyect*), todo lo referente a los objetos está situado en las carpetas *Imported Assets* (en caso de haber sido importados) y *Objects* de forma ordenada. En caso de contener componentes de sonido, estas están situadas en la carpeta *Sounds*.

Personajes

- *Phiby Third Person Controller* (TCP): este es el personaje principal del juego, el cual el usuario maneja para moverse por el mundo abierto, la cámara asociada a este personaje es en tercera persona. Las dos únicas formas de control del personaje son las dos siguientes:
    - Control por teclado y ratón: se utilizan W, A y D para avanzar, ir hacia la izquierda e ir hacia la derecha respectivamente, al mantener pulsado el botón *Shift* la velocidad del personaje aumenta (*Sprint*), al pulsar una vez la tecla *Space*, Phiby salta.
    - Control mediante *Kinect Asset*: esta es una de las partes a las que se ha acudido en la integración de otros proyectos, esta parte consiste en generar los movimientos laterales que antes se utilizaban con A y D mediante rotaciones corporales hacia la izquierda y la derecha y el movimiento de avance y *sprint* mediante una rotación hacia adelante (mediante el sistema de ángulos mínimos [21].
- *Non-player Characters* (*NPC*): personajes secundarios. Estos personajes pueden disponer de múltiples funcionalidades, entre las cuales se encuentran la interacción con Phiby, con el entorno y con otros personajes NPC. La característica principal que presentan es que no son jugables por el Usuario.

En los directorios del proyecto mencionados en el apartado anterior (*Proyect*), todo lo referente a los personajes está situado en la carpeta *Characters* de forma ordenada. En caso de contener componentes de sonido, estas estan situadas en la carpeta *Sounds*.

Game Manager

Se trata del *Game Object* que contiene la componente **gameManager.cs**. Aunque la función actual de este *Script* se explica posteriormente, el motivo principal por el que se idea es para gobernar el resto de parámetros de forma jerárquica en futuras líneas de trabajo, especialmente para gobernar las funciones más arraigadas a la lógica principal del juego, es decir, el desarrollo de la historia de *Phiby´s Adventure 3D*.

Puesto que el *Game Manager* se crea directamente desde la jerarquía (*Hierarchy*) en la escena, este no aparece en los directorios del proyecto mencionados en el apartado anterior (*Proyect*). Sin embargo, el *Script* **gameManager.cs** estará situado en la carpeta *Game Logic* de forma ordenada.

Exercise Manager

Se trata del *Game Object* que contiene la clase **exerciseManager.cs** (creada por Daniel Iglesias) y sirve fundamentalmente para obtener los parámetros de configuración de un ejercicio establecidos en la plataforma web. Para ello, se requieren los siguientes datos (Figura 54):

- ID del ejercicio (*Integer*)
- Código del ejercicio (*String*)
- Nombre del ejercicio (*String*)

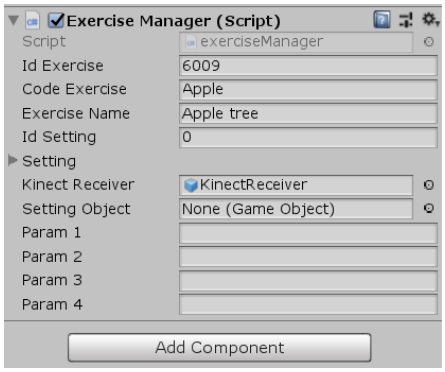


Figura 54. Componente exerciseManager.cs con sus variables públicas en el Inspector

El **exerciseManager.cs** primero intenta acceder a los parámetros concretos introducidos por el terapeuta en la web a través del ID, el código y del nombre del ejercicio. En caso de que este ejercicio exista en la web, guarda los parámetros del ejercicio en variables tipo *String*, para posteriormente poder ser utilizadas en Unity. Por otro lado, se beneficia de un método de referencia para enviar datos de resultados del ejercicio. Esta componente aparece a partir del momento en el que se carga la escena *islandScene*, y se ejecuta en cada escena en la que se vaya a realizar un ejercicio. Esta componente recibe los parámetros de configuración del ejercicio según el ID, el código y el nombre del ejercicio que se hayan instanciado en el inspector. En caso de no haber conexión con la web, el **exerciseManager.cs** acude a una plantilla por defecto, la cual se explicará detalladamente en el capítulo 3.4.3 “Plantilla por defecto sin configuración”.

Puesto que el objeto *Exercise Manager* se crea directamente desde la jerarquía (*Hierarchy*) en la escena, este no aparece en los directorios del proyecto mencionados en el apartado anterior (*Proyect*). Sin embargo, el *Script* **exerciseManager.cs** estará situado en la carpeta *Game Logic* de forma ordenada.

Componentes no destructibles

- ***Setting Sustain*** (*Game Object* y *Script*): se trata de un objeto con componente asociada no destructiva (es decir, el objeto permanece al cargar una nueva escena), por ello denominada "*Sustain"*. La clase **settingSustain.cs** (creada por Daniel Iglesias) guarda un diccionario con todos los parámetros de los ejercicios recogidos de la web Blexer-Med por los *Scripts* asociados al *Kinect Asset* que se explicarán posteriormente. Puesto que el *Setting Sustain* se crea directamente desde la jerarquía (*Hierarchy*) en la escena, este no aparece en los directorios del proyecto mencionados en el apartado anterior (*Proyect*). Sin embargo, el script **settingSustain.cs** estará situado en el subdirectorio *Kinect Asset* de la carpeta *Character*.

- ***Game Setting Sustain*** (*Game Object* y *Script*): al igual que el *Setting Sustain*, se trata de un objeto con componente asociada no destructiva. Esta componente se encarga de guardar todos **los parámetros del juego** en memoria, permitiendo mantener los cambios durante los saltos de escena. Esta componente se explica de forma más detallada en otro apartado. Puesto que el *Game Setting Sustain* se crea directamente desde la jerarquía (*Hierarchy*) en la escena, este no aparece en los directorios del proyecto mencionados en el apartado anterior (*Proyect*). Sin embargo, el *Script* **gameSettingSustain.cs** estará situado en el *Game Logic* de forma ordenada.

- ***Game Parameters Template*** *(Game Object y Script)*: plantilla con parámetros por defecto, se desarrolla la explicación sobre este objeto en el apartado 3.4.3.

Comunicación *Kinect Asse*t con Blexer-Med

Esta área de trabajo ha sido la base del proyecto en relación con la comunicación con K2UM para la captura de los movimientos corporales mediante la Kinect 2.0 y K2UM 2.0 para la comunicación entre Unity 3D y Blexer-Med. En este sector están implicadas las siguientes componentes: *Launcher.cs, Rotation.cs, UDPReceive.cs, UDPSend.cs, HeadersUDP.cs, SettingManager.cs.* En el **Anexo I** es posible consultar las funcionalidades de estas componentes, extraídas a partir de la documentación y comentarios creados por César Luaces [3] y actualizados por Daniel Iglesias. En los directorios del proyecto mencionados en el apartado anterior (*Proyect*), todo lo referente al *Kinect Asset* estará situado en el subdirectorio *Kinect Asset* de la carpeta *Character*.

3.3. Cambios entre escenas

3.3.1. Interacción entre el personaje principal y otros objetos 3D

Como se ha mencionado en apartados anteriores, la interacción entre el personaje principal y otros objetos 3D en el mapa está presente de forma reiterada en el juego gracias a que estos objetos son dinámicos.

En esta sección se describen otros objetos dinámicos necesarios para el cambio de escena a los juegos en los que el paciente realiza los ejercicios integrados previamente, sujetos a los ajustes proporcionados por el terapeuta desde la web. Visto que los cambios de escena a realizar serán "*Climb the Apple Tree*" y "*Chop the Wood*", los objetos utilizados como vínculo entre el mundo abierto y las dos escenas serán los mostrados en la Figura 55: un tipo determinado de árbol y un montón de troncos, que pueden estar ubicados en cualquier parte de la isla y de forma repetida.

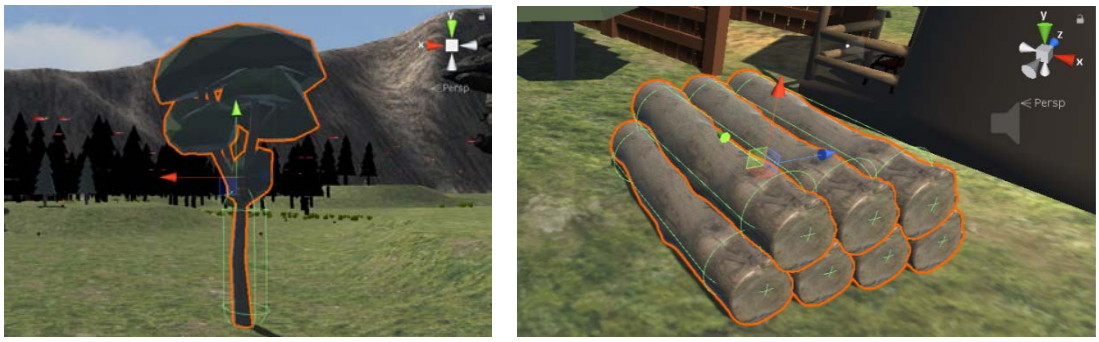


Figura 55. Objetos utilizados para el cambio de escena

Estos objetos, entre otros componentes, tienen un *collider* asociado el cual se utiliza como *Trigger* para ser capaz de manejar el enlace entre ambas escenas. De esta forma, se determina cuándo un objeto se encuentra abierto al cambio de escena (*Is Trigger* activo) *y* cuándo este se encuentra cerrado (*Is Trigger* inactivo, se produce una colisión). En la Figura 56 se muestra un *Collider* en estado activo (*Is Trigger* marcado). Para realizar estos cambios, se requiere de un *Script* de control del objeto dinámico.

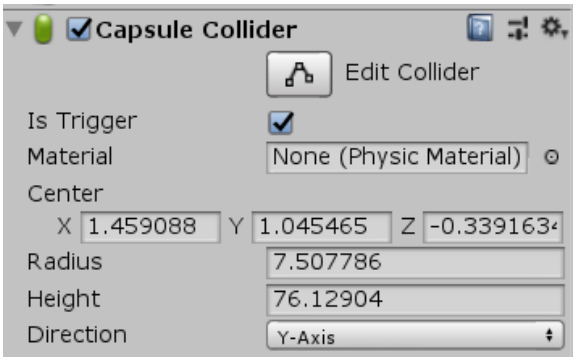


Figura 56. *Collider* con el parámetro *Is Trigger* activo útil para el cambio de escena

Para este propósito, se precisa de antemano de la modificación de los ajustes del proyecto para ser capaces de realizar un cambio de escena de forma correcta. Para ello, es necesario añadir estas escenas a la ventana de escenas de construcción (Figura 57).

Una vez realizado este ajuste, se podrán establecer los cambios de escena para las escenas añadidas al *Build Settings*. El número que se asigna a cada escena seleccionada es útil para los saltos de escena descritos, puesto que se puede acudir a ellos para iniciar la escena asociada a este. Otra forma de iniciar la escena es invocarla utilizando el nombre de esta.

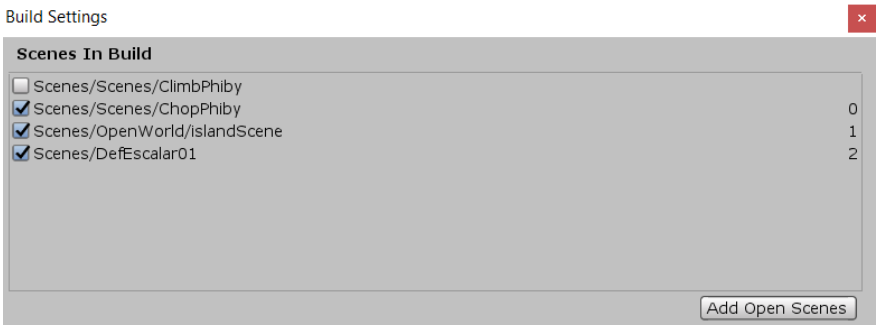


Figura 57. Ventana *Build Settings* con las escenas añadidas

Esta ventana también dispone de una sección para poder ejecutar el videojuego de *Phiby´s Adventure 3D* en cualquier plataforma. En nuestro caso, un PC de *Windows*, se añadirá la plataforma por defecto para el este: *PC, Mac & Linux Standalone*. De este modo, como se puede ver en la parte derecha de la ventana de *Build Settings* de la Figura 58, se seleccionará en Unity 3D la plataforma en la que se será ejecutado. En caso contrario, en el supuesto de que se intente acceder a otra plataforma, sería necesario descargar las componentes de este en el motor. El área de la ventana de *Build Settings*, aunque se valora como un elemento que es fundamental comprender y en un futuro será necesario para construir el ejecutable del juego completo, por el momento sólo sirve para generar cambios de escena.

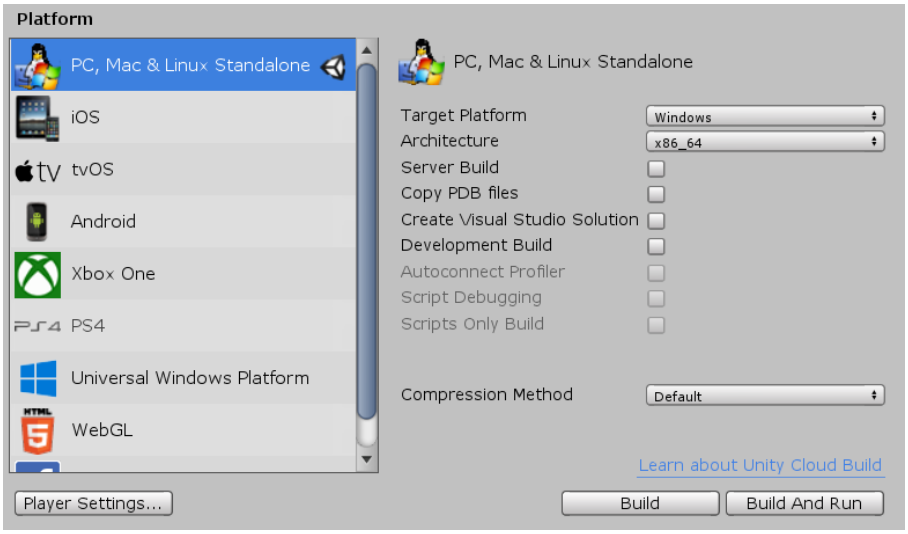


Figura 58. Ventana *Build Settings* con la plataforma seleccionada.

Una vez estudiado el panel de construcción de escenas, el proyecto ya está dispuesto para la composición de cambios entre ellas. Para esto, a través de los *Scripts* asociados a los objetos dinámicos mencionados anteriormente, Unity 3D nos permitirá realizar los cambios de escena que se consideren oportunos.

No obstante, para entender de forma más clara el funcionamiento de ambos *Scripts* que definen el comportamiento del objeto, se ha creado el diagrama de la Figura 59 que

explica la parte del código que se encarga de la lógica para el cambio de escena. En caso de entrar en el *Trigger* (***onTriggerEnter.cs***) con el jugador principal, que tiene la etiqueta "*Player*", se destruye el *Kinect Receiver* mediante el acceso al **rotation.cs** y se carga la escena relacionada con el objeto. Si el objeto es un árbol para escalar como el que hemos visto previamente, se carga "*Climb the Apple Tree*", en caso de ser el conjunto de troncos, se carga "*Chop the Wood*". Por otra parte, es preciso mencionar que posteriormente se verán las condiciones de apertura, las cuales nos dejarán cambiar de escena (*Trigger* activo) o no (*Trigger* inactivo) según las circunstancias del entorno y del objeto dinámico concreto con el que interactúe el jugador.

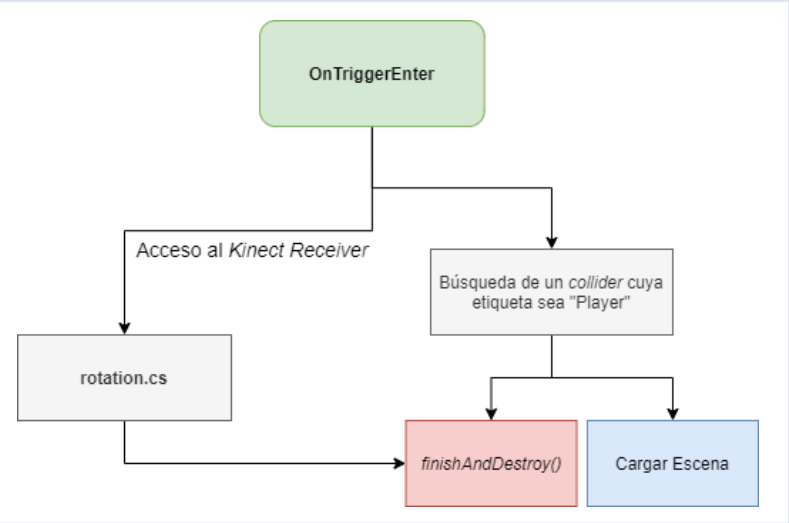


Figura 59. Funcionamiento del *Script* asociado al objeto para el cambio de escena

3.3.2. Conservación de parámetros en los cambios de escena

Puesto que los cambios de escena normalmente llevan implícitos la eliminación completa de los progresos obtenidos en la escena anterior hasta el instante en el que se cierra esta, se proporciona una solución funcional que permite la conservación de parámetros en los cambios de escena.

En este apartado es donde se explican las componentes **gameManager.cs** y **gameSettingSustain.cs**, asociadas a los objetos *Game Manage*r y *Game Setting Object* respectivamente. Para ello, antes de explicar de modo más detallado su funcionamiento, es necesario tener presentes las dependencias de estas componentes entre sí y con otras componentes dentro de la aplicación de Unity 3D.

El **gameManager.cs,** de forma previa al inicio de la escena del mundo abierto, es decir, en el *Awake()*, es el encargado de instanciar el **gameSettingSustain.cs** y el **gameParametersTemplate.cs** (en caso de que los respectivos objetos a los que estas componentes estén asociados sean nulos, es decir, no hayan sido previamente instanciados). Esta implementación implica, que, al inicio del juego en el mundo abierto, se crean estas componentes sólo una vez y, aunque se vuelva a llamar a la función *Awake()* del **gameManager.cs**, no se duplicarán. Este suceso se explica de forma más sencilla en el diagrama de la Figura 60. Por otro lado, cabe aclarar que el único instante en el que se llame a la función *Awake()* será cuando se vuelva de otra escena (un ejercicio) a la isla,

dado que el **gameManager.cs**, asociado al objeto *Game Manager*, se encuentra únicamente en la escena de la isla.

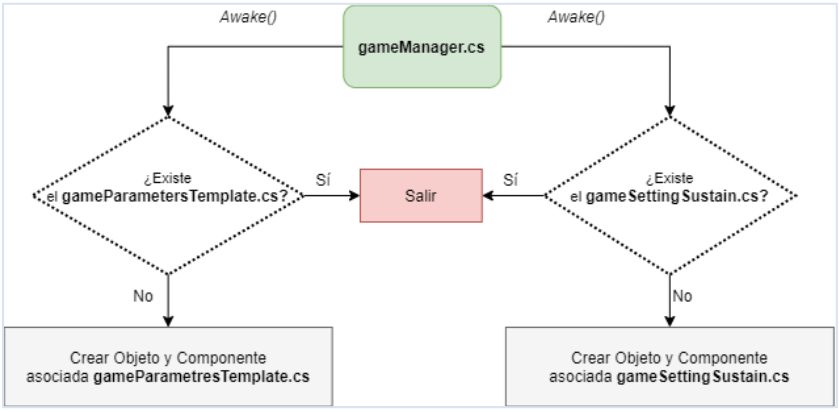


Figura 60. Diagrama de bloques de la creación de las componentes no destructibles del juego

Game Setting Object

Como se menciona en el apartado anterior, el *Game Setting Object* es un objeto no destructible, dado que el **gameSettingSustain.cs** asociado a este, contiene en sí el método *DontDestroyOnLoad(this)*. Este método evita que el objeto se destruya en los cambios de escena, por lo que esta información queda guardada en memoria en el intervalo de tiempo que el juego esté ejecutando. Esta componente aloja todas las variables que se necesiten compartir entre escenas y, además, es la encargada de abrir y cerrar objetos, es decir, estipula qué objetos dinámicos de cambios entre escenas deben estar abiertos al jugador principal para entrar en la escena asociada y en cuáles este no podrá entrar y, por tanto, colisionará, según se requiera en la parte de la historia del juego en la que Phiby se encuentre (condiciones de apertura). Asimismo, se tiene previsto la ampliación de este *Script* conforme se vayan añadiendo nuevos cambios de escena en el mapa, para el guardado en memoria de otras variables que se necesiten mantener durante la ejecución del juego.

Las variables guardadas por el momento en esta componente son las siguientes:

- **totalNumApples (int)**: Número total de manzanas que tiene el jugador recolectadas en el juego.
- **totalApplesNeed (int)**: Número total de manzanas que necesitas recolectar para el primer ejercicio.
- **treeAppleBig (boolean)**: Indica a la escena de *Climb the Apple Tree* si el árbol es grande o es pequeño (tiene más o menos manzanas).
- **treeOpen (array de booleans)**: Almacena si está abierto o cerrado un árbol según su ID.
- **currentTreeId (int)**: Identificador del árbol actual para su uso en la apertura o cierre de este.
- **totalNumOfMeters (int)**: Contador del número total de metros ascendidos entre todos los árboles.
- **lastScene (String)**: Comunica de qué escena acaba de salir Phiby.

La idea principal de este *Script* consiste en que se guarden todas la variables que se quieran mantener aquí en trabajos futuros tras la anexión de nuevos ejercicios a este.

Game Manager

Una vez indicadas las dependencias, se procede a explicar el funcionamiento de la componente **gameManager.cs** asociada al objeto *Game Manager*. Como se puede ver en el diagrama de la Figura 61, en el inicio de la escena de la isla, esta componente accede al *Script* **exerciseManager.cs** a partir del objeto *Exercise Manager*, encontrado para la exportación de los parámetros obtenidos por la web, o al *Script* **gameSettingSustain.cs,** a partir del objeto *Game Setting Object*, encontrado para la importación de los parámetros de condición para superar los ejercicios en este. Las fórmulas utilizadas para la generación de resultados condicionantes para el juego de la escena se clarifican en la sección 3.4.2 "Utilidad de parámetros de la web aplicados al juego" en la que se explican cuáles son las condiciones para superar cada escena.

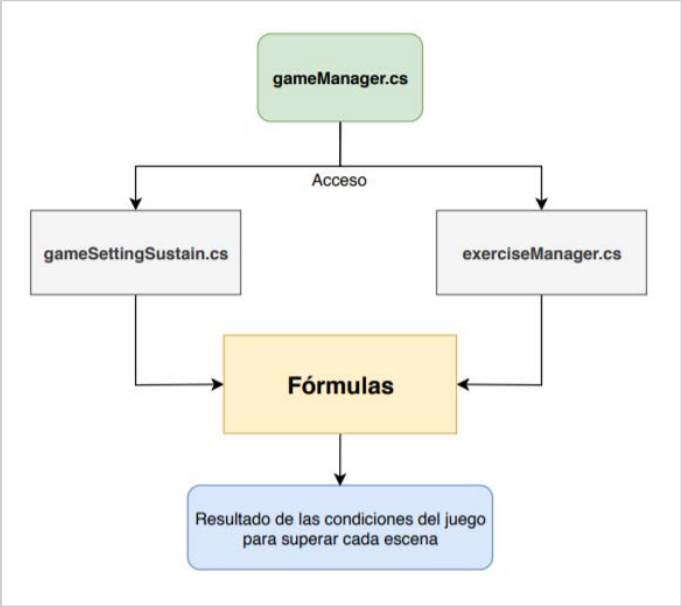


Figura 61. Funcionamiento del gameManager.cs

Asimismo, como se ha mencionado anteriormente, esta componente tiene como objetivo la función de manejador de todos los parámetros que estén relacionados con la lógica del videojuego para futuras líneas del proyecto de *Phiby´s Adventure 3D*.

3.3.3. Escena *Climb the Apple Tree*

Tras la integración del resto de proyectos en el mundo abierto, se comienza la interacción entre Phiby y el árbol de manzanas mediante un cambio de escena. Por lo que se procede a explicar cuál es la función de este árbol en el juego y cómo se han ampliado la lógica y las mecánicas de juego.

***Script* asociado al objeto**

En un paso previo a la entrada en la escena, se encuentra el *Script* asociado al objeto (en este caso el árbol), el cual es responsable de que este sea un objeto dinámico necesario para el cambio de escena. Además, este *Script* tiene otra peculiaridad que se ha implementado para lograr una mejor interacción entre el paciente y el juego. Esta

característica es la que define si el árbol va a ser grande o pequeño, mediante una variable pública de tipo *bool* llamada *bigTree* que, en caso de ser *true* hace que el árbol sea grande y en caso de estar en *false* sea pequeño. Por otro lado, el árbol tiene una variable pública de tipo número entero (*int*) llamada *treeID*, la cual, a través del Inspector, hace que cada árbol del mapa sea único.

En este caso, e booleano mencionado, en el momento en el que el jugador atraviesa el *Trigger* (*onTriggerEnter*), se añade el ID del árbol a la variable *currentTreeId* y el booleano *bigTree* a la variable booleana *treeAppleBig* del **gameSettingSustain.cs**. Una vez dentro de la escena, estas servirán para saber si el árbol es grande o pequeño y, por tanto, los metros que se recorrerán al trepar. El funcionamiento de esto último se mencionará con detalle en capítulos posteriores, y será útil para que el jugador pueda elegir qué árboles trepar en cada momento y para cumplir las condiciones que dicte el terapeuta (que estarán camufladas por requisitos que dicta el juego acorde con estas condiciones).

Además, algunos de estos árboles estarán cerrados siempre, como parte del entorno. Esta solución se aplica de forma sencilla, ya que todos los árboles que tengan como ID el 0, serán árboles decorativos que permanecerán cerrados durante toda la partida, al margen de cualquier condición. Esto también da la posibilidad de programar el juego para que, cada vez que se ejecute este, serán otros los árboles activos.

Entrada y salida

La entrada en el ejercicio propuesto, como se ha mencionado anteriormente, se realiza mediante un cambio de escena, el cual requiere tanto el guardado de las variables estables del juego como la destrucción del objeto con las componentes del *Kinect Receiver* asociadas a este. De esta manera, dentro de la nueva escena, el *middleware* se encontrará con otro objeto *Phiby Character*, el cual tendrá unas nuevas componentes *Kinect Asset* asociadas para la nueva conexión con el *middleware*. La destrucción de este objeto se invoca desde el método *finishAndDestroy()*, que contiene la componente **rotation.cs** vinculada al *Kinect Receiver* y se llama desde el objeto que carga la escena *Climb the Apple Tree*, en este caso, el árbol previamente mencionado en el apartado de “cambio de escena”.

Como se muestra en el diagrama de la Figura 62, la entrada y salida en la escena viene requerida por las condiciones de apertura que se explicarán justamente después. Tras cumplir estas y verificar que el árbol se encuentra abierto, se produce la entrada en la escena, donde Phiby tiene el objetivo de escalar un árbol para obtener manzanas. Una vez obtenidas, se produce el cierre del árbol concreto y la salida del ejercicio para volver a cargar la isla del mundo abierto. De ella, se registran el número de movimientos, el cual es equivalente al número de metros que escala el personaje y el número de manzanas recolectadas en el **gameSettingSustain.cs**. Próximamente, estas variables serán útiles para el cumplimiento de las condiciones dictadas por el terapeuta al juego.

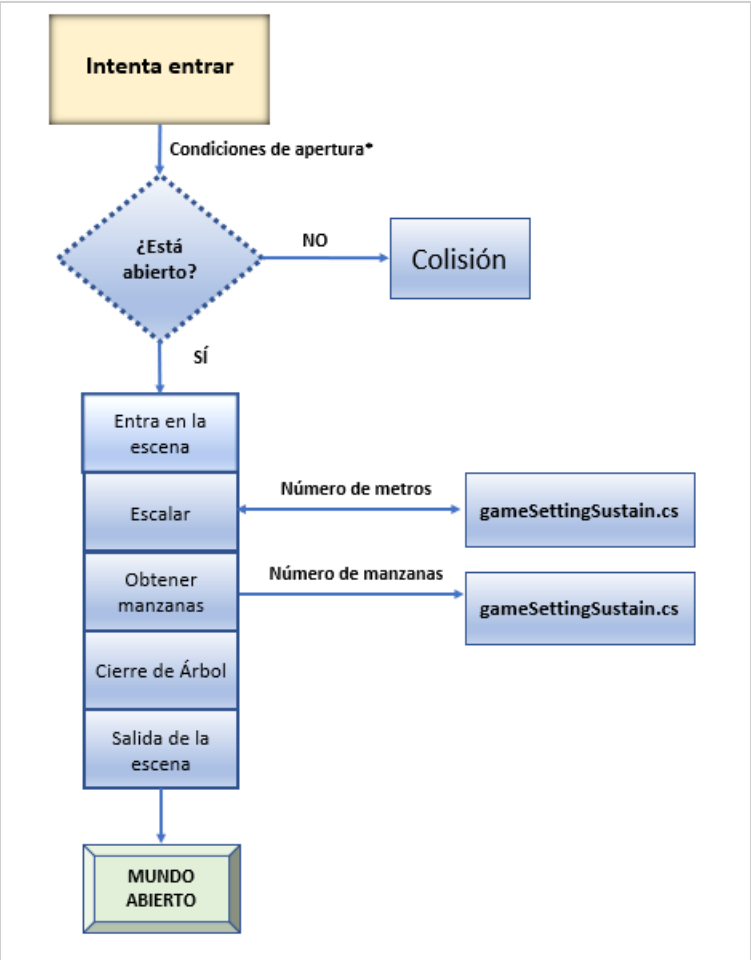


Figura 62. Entrada, recorrido y salida de escena

Operación de parámetros en la escena

Una vez explicada la línea temporal de la entrada y la salida de la escena, se procede a justificar la operación de parámetros en la escena para la obtención de manzanas y la resolución de las condiciones de salida de esta. Puesto que esta escena ha sido integrada de otro proyecto [3], el progreso se ha realizado a partir de este.

En primer instante se ha generado un contador de manzanas, al cual a partir del método *Random.Range(float min, float max)*, se le suman 1 o 2 manzanas de forma aleatoria en el **ClimbUp.cs** y se invocan desde el **ChangeTextClimb.cs** para, al final del ejercicio, enviar esta información al **gameSettingSustain.cs**. En este mismo instante, al haberse acabado la escena completamente, también se accede a esta componente no destructible para, desde ella, conmutar a *false* la variable del *treeOpen[*gameSettingSustain.*currentId]* que comunica al árbol donde el personaje principal ha terminado la escena, que este debe cerrarse, porque, de esta manera, se representa en el juego que ya se ha quedado sin manzanas y estas tienen que volver a crecer.

Paralelamente, se modifica el *Canvas* de la escena para que el jugador pueda visualizar en tiempo real cuántas manzanas lleva recogidas y se modificar las esferas de

partículas por modelos de manzanas 3D. Del mismo modo, se acceden a los parámetros del **exerciseManager.cs** para servirse de ellos en la ejecución de las condiciones de la escena. Por un lado, de nuevo accediendo al **gameSettingSustain.cs** desde el **ChangeTextClimb.cs**, se revisa si el árbol en el que nos encontramos es grande o pequeño, para así calcular el **máximo número de metros** que se pueden realizar en este. Los casos en los que la escena termina son los siguientes:

1. En el caso en el que el número de metros alcanzado por el jugador sea igual o mayor que el **máximo número de metros** que se pueden realizar en la escena (como se ha dicho antes, este número variará en función de si el árbol es grande o pequeño).
2. En el caso de que el **número de movimientos totales** entre la suma de todos los ejercicios en los múltiples árboles recogido del **gameSettingSustain.cs** sea al menos igual al **número de movimientos máximo** aportado como parámetro de la web (*num_mov_max*) que el paciente puede realizar antes del tiempo de descanso.
3. En el caso en el que el **tiempo máximo** aportado como parámetro de la web (*tiempo_max*) que el paciente puede realizar en la escena actual total exceda el tiempo transcurrido en el ejercicio, siempre que se haya realizado el número mínimo de movimientos en este.

Véase en la Figura 63.

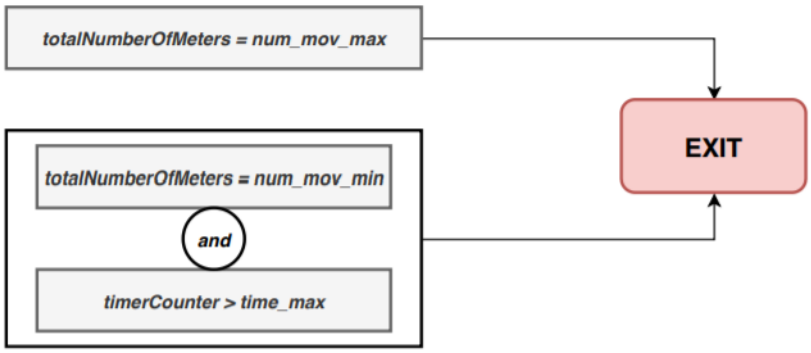


Figura 63. Casos en los que termina la escena

En estos dos últimos casos, puesto que se ha llegado al límite máximo impuesto por el terapeuta, es requerido un tiempo mínimo de descanso obligatorio, por lo que se cierran todos los árboles presentes en el mundo abierto hasta nueva orden.

Por otro lado, se implementan operadores de generación de tamaño del árbol en la escena a partir del booleano *treeAppleBig* el cual se exporta del **gameSettingSustain.cs** el cual se explica en la Figura 64.

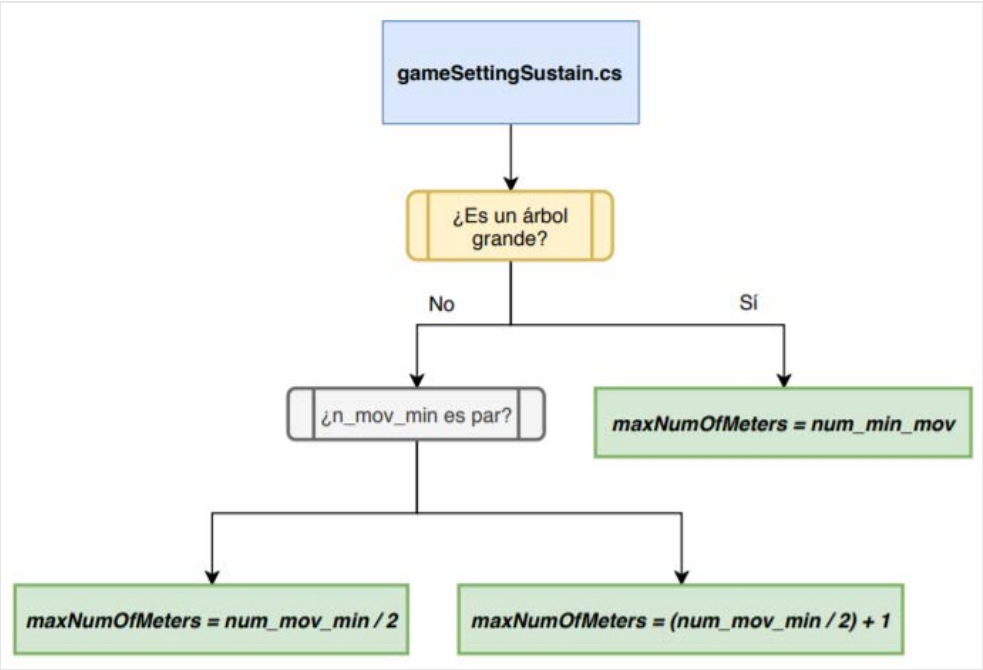


Figura 64. Operadores utilizados en el interior de la escena para el número mínimo de movimientos a realizar

Este modo en el que se generan los operadores que condicionan de forma imperativa a cumplir el mínimo número de movimientos que debe realizar el usuario. De esta manera, el jugador tiene libertad de elección entre los diferentes árboles que se encuentra en la isla, pero, por otra parte, sin ser necesariamente consciente de ello, realiza los movimientos dentro del rango aplicado por el terapeuta desde la web, dado que el usuario se centra exclusivamente en el número de manzanas a recoger, más que en el número de repeticiones que debe realizar durante la terapia, por lo que se genera mayor inmersión de este en el juego.

Condiciones de apertura

Para no sobrecargar un paciente durante el juego, es decir, para evitar que supere un número máximo de movimientos, quizás debido a muchos fallos que le llevan a querer repetir el ejercicio del árbol de manzanas, se implementarán mecanismos de descanso. Uno de estos es la barra de energía que muestra al jugador cuando debe ir a recuperarla. El plan de desarrollo a futuro que se está ideando en el proyecto es la recogida de setas por la isla como método de entretenimiento para el paciente en el que se pueda recuperar la energía al completo, siempre que, sin este ser consciente de la situación, se haya recogido al menos una seta y haya transcurrido el tiempo de descanso, como se muestra en la Figura 65. En el momento que haya recuperado la energía, los arboles serán accesibles otra vez.

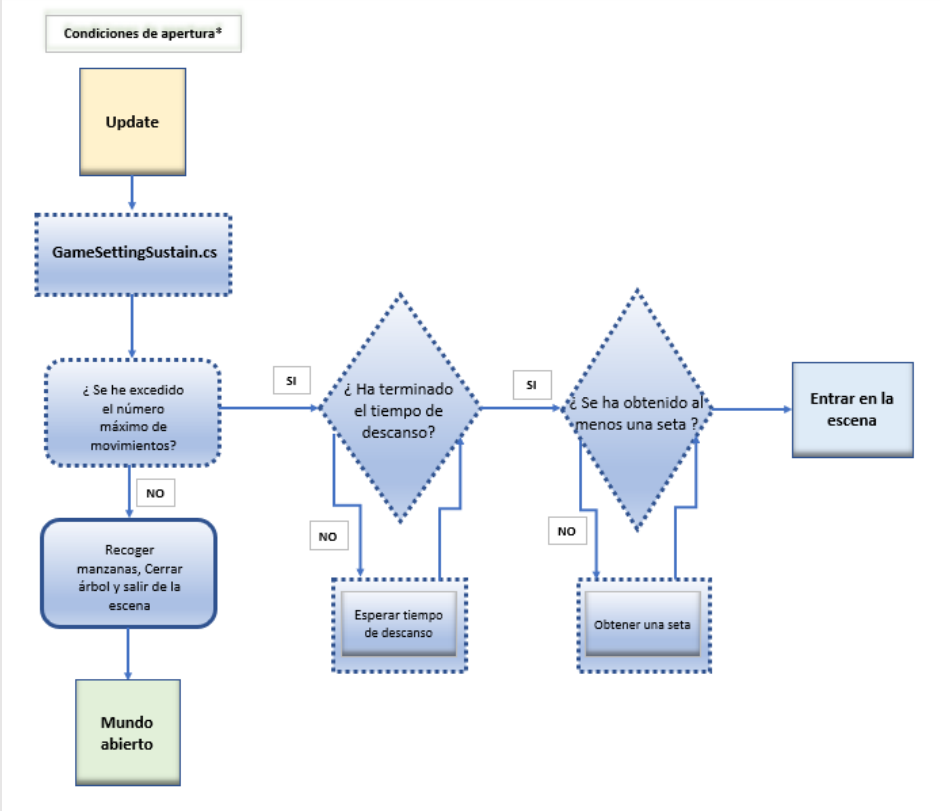


Figura 65. Condiciones de apertura I

Por otra parte, una condición imprescindible para que el personaje pueda entrar en la escena, es que no se haya realizado esta acción en ese árbol anteriormente. De esta manera, por una parte, el juego es más realista puesto que se han agotado las manzanas que había en ese árbol y además así el usuario tiene posibilidad de buscar en otros lugares de la isla. Un ejemplo simplificado de esta condición se muestra en el diagrama de la Figura 66.

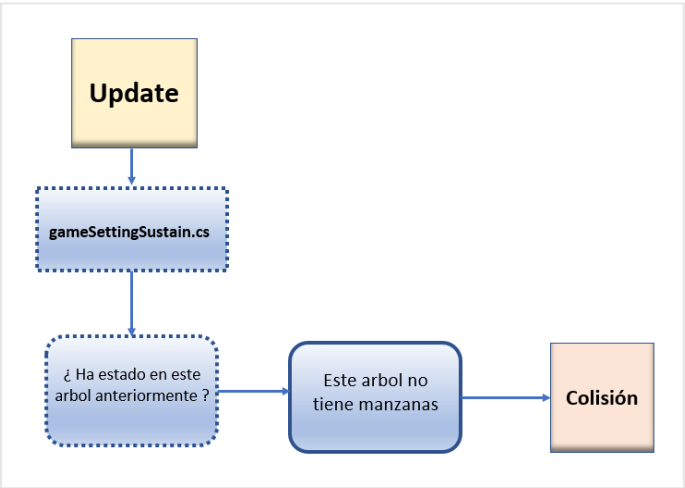


Figura 66. Condiciones de apertura II

3.4. Ejercicios configurables a distancia

Este punto se centra en la configuración de parámetros que interfiere en el juego a través de internet, mediante la web. Para ello, se explica de forma detallada cuáles son las vías de avance en el juego *Phiby´s Adventure 3D* y en qué partes de este acometen los parámetros proporcionados por la web Blexer-Med a través del *middleware* K2UM 2.0.

3.4.1. Posibilidades de configuración que ofrece la web

Como se ha mencionado en la sección de antecedentes de este PFG, la web Blexer-Med 2.0 tiene implementadas una gran variedad de funcionalidades que serán útiles para la organización entre los diferentes centros en los que se pretenda realizar este tipo de rehabilitación con acceso a ella. Sin embargo, es evidente que, no sólo será útil como medio entre los diferentes centros, sino que también será necesaria para la configuración de los múltiples juegos adaptados a ella.

Para cualquier juego, una de las facultades desarrolladas en esta web es la creación de ejercicios por el super administrador, estos ejercicios aparecerán como parte de un juego concreto (en este caso, *Phiby´s Adventure 3D*) y serán manejados por el terapeuta en la sección *Clinician* de la plataforma. En la Figura 67 se muestra un ejemplo del paciente "Juan Test" en el que el terapeuta ha seleccionado el juego *Phiby´s Adventure 3D* y puede ver el ID, el Alias, el título y la descripción de los ejercicios de este juego.

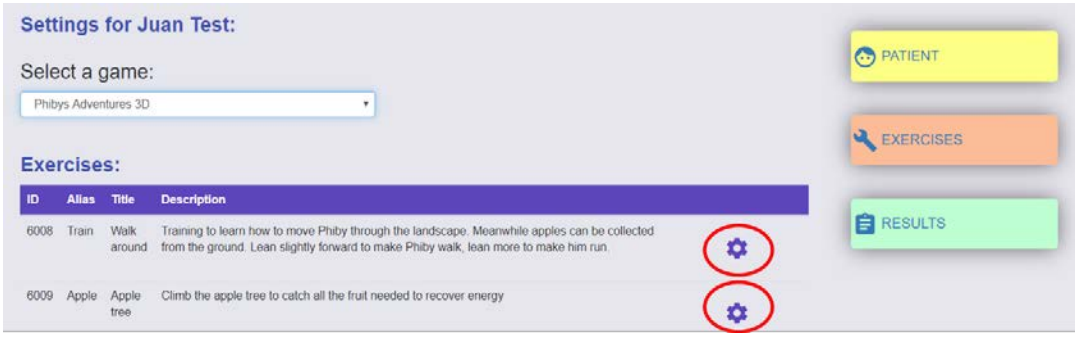


Figura 67. Ejemplo de la estructura de ejercicios juego para un paciente seleccionado

Después, el terapeuta elije el ejercicio al que quiere añadir los parámetros pulsando la rueda dentada marcada con una circunferencia roja en el ejemplo de la figura anterior y se abre una ventana donde aparece el cuadro mostrado en la Figura 68.

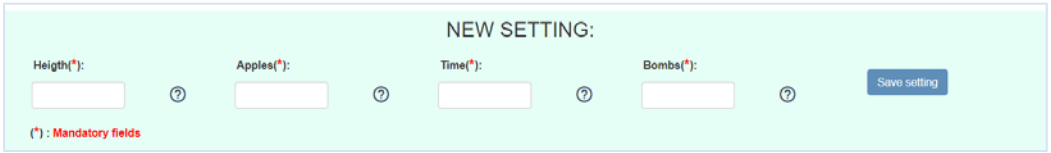


Figura 68. Cuadro de configuración de parámetros para el paciente por el terapeuta

Una vez añadidos los parámetros para el paciente, estos quedan almacenados en una base de datos. Asimismo, se puede disponer de varias configuraciones y, de esta manera, el terapeuta puede decidir de forma intuitiva el cambio de parámetros instantáneamente (Figura 69).

ID	Signature	Date	Heigth	Apples	Time	Bombs	Comment	In Use
7554	Juan Alberto Garcia	29/10/2019 12:05	10	30	10	50		
7551	Juan Alberto Garcia	10/10/2019 15:30	10	12	1	1		
7550	Juan Alberto Garcia	10/10/2019 15:28	10	30	10	10		

Figura 69. Ejemplo de diferentes configuraciones para un ejercicio

Los ID (número de identificación) se generan automáticamente para cada uno de los parámetros de la web, se crean de forma automática a partir de un prefijo (unidad de millar del ID):

- ID para los administradores a partir del 500.
- ID para los centros a partir del 1000.
- ID para los terapeutas a partir del 2000.
- ID para los pacientes a partir del 3000.
- ID para los juegos a partir del 5000.
- ID para los ejercicios a partir del 6000.
- ID para los ejercicios a partir del 7000.

En el desarrollo de la solución para este PFG, serán necesarios únicamente el usuario y contraseña de un paciente y los id para el juego y los ejercicios del juego a emplear. Las posibilidades que la web ofrece son indispensables para el entendimiento del posterior desarrollo del juego y de los ejercicios a manipular. En el diagrama mostrado en la Figura 70 se representa de forma simplificada la comunicación entre el terapeuta y el motor Unity 3D donde se aloja el proyecto de *Phiby´s Adventure 3D*.

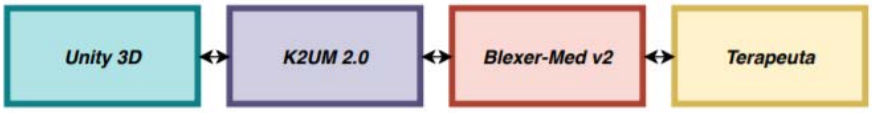


Figura 70. Diagrama de comunicación entre el terapeuta y el juego

3.4.2. Utilidad de los parámetros de la web aplicados al juego

Climb the Apple Tree

Como se ha podido ver anteriormente, el **gameManager.cs** accede al **exerciseManager.cs** y, a través de las fórmulas de condicionamiento del juego, importa en el **gameSettingSustain.cs** los resultados obtenidos. Pero antes, es importante tener en cuenta cuáles son los parámetros obtenidos de la web. Esta permite cuatro parámetros por ejercicio, de los cuales se han definido tres.

- Parámetro 1 (*num_mov_min*): es el número de movimientos mínimo que el paciente debe realizar en total durante el ejercicio completo.
- Parámetro 2 (*num_mov_max*): es el número de movimientos máximo que el paciente puede realizar en total durante el ejercicio completo.
- Parámetro 3 *(tiempo_max*): es el número máximo de segundos que puede estar el paciente haciendo el ejercicio seguidamente.

Estos parámetros, no sólo son aplicados al juego como condicionantes del ejercicio *Climb the Apple Tree*, sino que también son utilizados para la obtención de parámetros en la historia de *Phiby´s Adventure 3D*. La complejidad reside en conseguir que todas las condiciones aplicadas al juego se den de forma significativa equitativamente, es decir, que ninguno de ellos se vea determinado por otros para llevarse a cabo.

Para la aplicación de las fórmulas asociadas a los parámetros extraídos, se aplica el método *Random.Range(float min, float max)* el cual devuelve un número tipo *float* entre el mínimo (incluido) y el máximo (no incluido). Este método se aplica al resolver la fórmula que devuelve el número de manzanas que necesita el juego (en este caso, la abuela) para finalizar el ejercicio. La fórmula para este número de manzanas, siempre devuelve un resultado diferente acotado entre el número mínimo de movimientos y el doble del mínimo de movimientos establecidos por el terapeuta:

$$num\_apples = (int)\ Random(num\_mov\_\min + 1f\ , (2 * num\_mov\_\min)\ + 1f)$$

Posteriormente, este valor se guarda en el **gameSettingSustain.cs** para evitar que se destruya el cálculo extraído en los cambios de escena. Dado que en las escenas de *Climb the Apple Tree* se recogen manzanas y se guardan en la variable **totalNumApples** de la componente del *Game Setting Object*, mediante una sencilla línea de código se puede avisar al jugador si ya ha llegado al número de manzanas mínimo requerido o si, en cambio, este debe seguir recogiéndolas.

Como se puede apreciar en la fórmula, al *num_mov_min* y al *2*num_mov_min* se les suma 1. Esto se realiza para conseguir que todas las condiciones mostradas en el apartado anterior sobre la escena se cumplan antes de que se consigan todas las manzanas para acabar el ejercicio. En caso de no sumar las variables, el resultado habría sido:

$$num\_mov\_\min \leq \boldsymbol{num\_apples} < 2 * num\_mov\_min$$

En este caso, al sumar 1 a ambas variables, se pretende que el valor de *num_apples* se mantenga acotado de la siguiente manera:

$$num\_mov\_\min < \boldsymbol{num\_apples} \leq 2 * num\_mov\_min$$

Se requiere acotar este valor de esa forma para conseguir que el jugador realice el número de movimientos mínimo antes de completar la escena, esto es, antes de que consiga todas las manzanas requeridas (***num_apples***).

Todas las manzanas obtenidas para este ejercicio se pueden recoger únicamente en la escena *Climb the Apple Tree*, por lo que, las manzanas repartidas por el mundo abierto, no servirán para este ejercicio. La idea de estas últimas es cambiarles el modelo del objeto o, en otras palabras, utilizar otro tipo de alimento (como setas, por ejemplo) para que, al recogerlas, se aumente la energía para poder volver al ejercicio después de haber pasado el tiempo máximo recibido como parámetro desde la web. En el ejemplo de la Figura 71 se muestran una serie de parámetros introducidos desde la web y la salida tras pasar por el **gameManager.cs** para cada jugador.

num_mov_min	2*num_mov_min	num_apples jugador 1	num_apples jugador 2
5	10	8	8
7	14	9	9
10	20	15	17
13	26	15	21
15	30	20	27
20	40	30	26

Figura 71. Resultados del número de manzanas requerido para dos jugadores distintos

Como se puede apreciar, a pesar de que el número mínimo y doble del mínimo de movimientos es el mismo, el número de manzanas que debe obtener es diferente, acotado siempre por el terapeuta según el número mínimo de movimientos que puede realizar.

Una cuestión a tener en cuenta, es el caso en el que el terapeuta escriba dos números de forma incorrecta, esto es, si introduce un número de movimientos mínimo mayor que el número de movimientos máximo. El plan inicial para estos casos es que, desde la web, se implemente una sección donde no se le permitan incongruencias de este tipo o de otros como escribir letras en lugar de números. Asimismo, estas fórmulas se deciden así para permitir que se cumplan todas las condiciones proporcionadas por la web antes de que la parte de la historia del juego donde se involucran los parámetros termine.

Por último, se explica el reparto n_mov_min entre los árboles activos para proporcionar diferentes posibilidades:

- Dos árboles “bajos” con **n_mov_min/2 = [n_mov_min/2, n_mov_min]** manzanas
- Un árbol “alto” con **n_mov_min = [n_mov_min, 2*n_mov_min]** manzanas.

De esta forma, el usuario nunca va a realizar menos del número mínimo de movimientos, puesto que se necesitan como mínimo dos árboles bajos para conseguir el reto de las manzanas y, en caso de que el usuario llegue al máximo número de movimientos, como se ha mencionado con anterioridad, se termina la partida y se inicia el proceso de descanso.

3.4.3. Plantilla por defecto sin configuración

Hasta ahora se han utilizado los parámetros de la web aplicados al juego, teniendo en cuenta que el usuario puede acceder a esta a través de internet y que, asimismo, está registrado en esta como paciente. En este apartado, se expone una solución en caso de no cumplirse alguna de estas premisas.

En caso de que el usuario no tenga la alternativa de conectar su ordenador a internet o no esté registrado en esta, en el *middleware* K2UM 2.0, existe la opción de "Continuar sin cargar configuración" [16] como el que aparece en la Figura 72, la cual no requiere de nombre de usuario ni contraseña para servirse del programa.

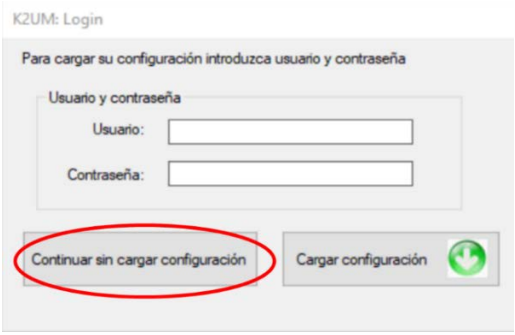


Figura 72. *Login* de acceso a la web como paciente [16]

En este caso, la aplicación de Unity 3D no recibirá los parámetros de configuración de la web y, por lo tanto, inicialmente estos parámetros serán nulos o estarán vacíos. Para el desarrollo de una solución consistente que permita al usuario jugar a *Phiby´s Adventure 3D*, en caso de cargar el *middleware* sin configuración, se invoca una plantilla prediseñada, editable y ampliable como resultado. Para mayor facilidad de edición de esta, la plantilla se genera como un *Script*, de forma que se puede acceder a ella fácilmente y por parte de los programadores del videojuego en el momento de añadir otras escenas nuevas en el mundo abierto. El diseño de esta plantilla, desde Unity 3D, tiene la siguiente forma (Figura 73)

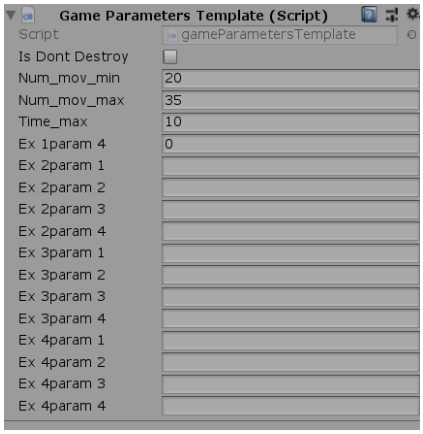


Figura 73. Plantilla de parámetros por defecto rellenable para cada escena del juego

Como se ha mencionado en capítulos anteriores, esta plantilla se genera en el juego una vez se ejecute por el **gameManager.cs** una única vez. Esta plantilla se crea como **Objeto no Destructible** (Plantilla en el **Anexo IV**). A partir de ese momento, a esta

platilla se accede desde el **exerciseManager.cs** en caso de que se haya cargado el *middleware* sin configuración, esto es, cuando el *Setting Object* sea nulo (Figura 74).

```
GameObject GameParametresTemplate = GameObject.Find("GameParametresTemplate");
gameParametresTemplate gameParametresTemp = GameParametresTemplate.GetComponent<gameParametresTemplate>();

 if (idExercise == 6009)
 {
     param1 = gameParametresTemp.num_mov_min;
     param2 = gameParametresTemp.num_mov_max;
     param3 = gameParametresTemp.time_max;
     param4 = gameParametresTemp.Ex1param4;
 }
 // Ejemplo
 if (idExercise == <ExerciseId>)
 {
     param1 = gameParametresTemp.Ex2param1;
     param2 = gameParametresTemp.Ex2param2;
     param3 = gameParametresTemp.Ex2param3;
     param4 = gameParametresTemp.Ex2param4;
  }
```

Figura 74. Sección del código que se encarga de la asignación de parámetros de la plantilla al exerciseManager.cs

Esta asignación de parámetros se realiza siempre que se ejecute el **exerciseManager.cs**, por lo que, en el momento en el cual el **gameManager.cs** u otra componente acceda a estos parámetros, estos serán los recogidos de la plantilla por defecto.

4. Pruebas básicas de funcionamiento

Se realizan las pruebas de funcionamiento después de la creación de una versión estable del proyecto. El propósito principal por el que se realizaron las pruebas de funcionamiento es comprobar el funcionamiento del algoritmo que se encarga de calcular las variables necesarias para el juego y comprobar que definitivamente los límites marcados por el terapeuta están correctamente implementados. Otro motivo, es comprobar la experiencia del usuario ante un proceso concreto del juego y cuál es el nivel de dificultad de adaptación de este.

Puesto que el juego aun está en fase de desarrollo, es decir, aun no está terminado, se puede averiguar también cuál es el resultado de los cambios de escena mediante la captura de movimientos con la Kinect 2.0. Por último, se les pide a los usuarios un *feedback* del juego para que este pueda expresar libremente qué podría mejorar en este.

4.1. Pruebas guiadas para un nivel

El objetivo de esta primera prueba es la comprobación del manejo del entorno por un usuario no experimentado. Se realiza un ejercicio que está fragmentado en dos partes diferenciadas. Para ello, se introduce al personaje principal controlado por la Kinect 2.0 cerca de la cabaña y el objetivo es, llevarlo hacia un árbol de manzanas, entrar en la escena, coger las manzanas y salir de la escena. Se recogen como resultado el tiempo total que necesita el jugador, el número de movimientos que realiza y el número de manzanas que obtiene. Este último valor es para comprobar si el algoritmo trabaja correctamente. Además, se pregunta a los voluntarios sobre la dificultad en el movimiento en el mundo abierto y dentro del ejercicio, lo cual deben evaluar de 1 (sin dificultad) a 10 (mucha dificultad).

Se realizan 6 muestras efectuadas por personas de edades entre 11 y 48 años que no tienen ningún problema físico y que no hayan probado el juego con anterioridad. La configuración del mínimo y máximo número de movimientos se ha mantenido constante en 15 y 30, dejando un tiempo máximo de 70 segundos para la realización del ejercicio. Los resultados obtenidos se presentan en la Figura 75. Para superar la prueba se requiere la obtención de entre 15 y 30 manzanas.

En los resultados obtenidos se pueden destacar:

Los jugadores 1 y 3 no superaron la prueba, ambos aparentemente por la misma razón:

- El **Jugador 1** decidió terminar de jugar antes de llegar al número mínimo de manzanas que tenía que obtener.
- El **Jugador 3** tuvo dificultades con el manejo del control de *Phiby* en el mundo abierto, por lo que le costaba llegar desde la posición inicial a la escena de escalar el árbol. Una vez llegó, realizó el ejercicio de forma sencilla con éxito, pero cuando volvió al mundo abierto, decidió finalizar el ejercicio.

- El resto de jugadores estaban de acuerdo en que, aunque el control del personaje principal es posible de manejar, durante la primera toma de contacto se necesitaba un pequeño periodo de tiempo para dominarlo.

- Otra deducción que se puede obtener a partir de las pruebas, es que el movimiento del Phiby durante la escena ha resultado sencillo para todos los jugadores presentes.

							OPEN WORLD	ESCENA					
JUGADOR	EDAD	Nivel	Número mínimo de movimientos	Número máximo de movimientos	Tiempo Máximo	¿Superó la prueba?	Dificultad en el movimiento (1-10)	Dificultad en el movimiento(1-10)	Número de árboles alcanzados	Tiempo total transcurrido	Número de movimientos realizados (METROS)	Número de manzanas obtenidas	Número de manzanas necesarias
1	15	1	15	30	70	No	5	1	1	136	7	11	21
2	12	1	15	30	70	Sí	4	1	2	87	14	20	24
3	11	1	15	30	70	No	8	1	1	150	7	10	22
4	47	1	15	30	70	Sí	5	1	1	90	15	23	23
5	48	1	15	30	70	Sí	4	2	3	180	24	33	33
6	48	1	15	30	70	Sí	7	2	2	180	14	21	21

Figura 75. Resultado de las pruebas básicas para un único nivel de dificultad

- Como se puede ver en la Figura anterior, hay dos resultados de "número de movimientos realizado" que aparecen en rojo, dado que se han quedado por debajo del número mínimo de movimientos introducido por el terapeuta. En este caso, la causa de este problema era un error en el código, en el chequeo del número máximo de movimientos permitidos por el terapeuta, que se visualiza en la Figura 76. El error ha sido la escritura de *maxNumOfMeters* en lugar de *n_mov_max* en el *if* enmarcado en rojo. La función de esta línea de código es comprobar si el número máximo de movimientos es par, en este caso se divide el número máximo de movimientos dentro de la escena entre dos para calcular la altura del árbol pequeño (y así los movimientos necesarios para terminarlo), en caso contrario, se divide entre dos y se suma uno.

```
int maxNumOfMeters = 0;
if (!gameSetSus.treeAppleBig)
{
    if ((maxNumOfMeters % 2) == 0)
    {
        maxNumOfMeters = n_mov_min / 2; //Small Tree
    }
    else
    {
        maxNumOfMeters = (n_mov_min / 2) + 1; //Small Tree (to make the user do the minimum number of movements.
    }
}
else
{
    maxNumOfMeters = n_mov_min; // Big Tree
}
/* END */
```

Figura 76. Error causado por cambio de variables

La solución propuesta es la mostrada en el Figura 77 mostrada a continuación

```
if ((n_mov_min % 2) == 0)
{
    maxNumOfMeters = n_mov_min / 2; //Small Tree
}
else
{
    maxNumOfMeters = (n_mov_min / 2) + 1; //Small Tree (to make the user do the minimum number of movements.
}
```

Figura 77. Solución propuesta al error causado

4.2. Pruebas guiadas por niveles

Tras la solución de los errores previamente mencionados, se procede a unas pruebas más exhaustivas con diferentes jugadores, donde se inicia el juego con diferentes parámetros aportados desde la web. En este caso se trata de 4 personas entre 25 y 61 años, ninguna de ellas participó en las pruebas anteriores.

Se han establecido los siguientes tres niveles:

- Nivel 1: entre 15 y 30 movimientos en un tiempo máximo de 30 segundos.
- Nivel 2: entre 15 y 30 movimientos en un tiempo máximo de 20 segundos.
- Nivel 3: entre 20 y 35 movimientos en un tiempo máximo de 10 segundos.

Para superar la prueba se requiere la obtención del número de manzanas necesarias. Los resultados se representan en la Figura 78.

							OPEN WORLD	ESCENA					
JUGADOR	EDAD	Nivel	Número mínimo de movimientos	Número máximo de movimientos	Tiempo Máximo en la escena	¿Superó la prueba?	Dificultad en el movimiento (1-10)	Dificultad en el movimiento(1-10)	Número de árboles alcanzados	Tiempo total transcurrido	Número de movimientos realizados (METROS)	Número de manzanas obtenidas	Número de manzanas necesarias
7	60	1	15	30	30	Sí	3	1	2	123	23	34	29
		2	15	30	20	Sí			2	100	23	34	25
		3	20	35	10	Sí			2	104	26	37	37
8	61	1	15	30	30	Sí	5	3	3	170	30	40	26
		2	15	30	20	Sí			1	65	16	26	25
		3	20	35	10	Sí			1	61	20	29	28
9	25	1	15	30	30	Sí	4,5	1	3	170	30	40	30
		2	15	30	20	Sí			2	67	16	27	19
		3	20	35	10	Sí			1	42	15	24	23
10	29	1	15	30	30	Sí	6	4	2	178	17	26	26
		2	15	30	20	Sí			2	86	17	37	30
		3	20	35	10	Sí			2	73	20	31	25

Figura 78. Resultado de las pruebas básicas para varios niveles de dificultad

En este caso, los resultados manifiestan un resultado bastante positivo para los diferentes niveles, dado que todos los participantes han conseguido superar las diferentes escenas realizando el número de movimientos requerido por el terapeuta en un tiempo inferior al máximo necesario en cada escena.

En cuanto a la dificultad de movimiento del jugador, como se expresa en ambas tablas dentro de este apartado, el resultado no se considera tan positivo como se desea, por lo que se requiere una mayor profundización del controlador mediante Kinect en futuras vías del proyecto de *Phiby´s Adventure 3D.*

Pruebas de funcionamiento a través de la plantilla

Los resultados obtenidos tras la utilización de la plantilla por defecto son acertados en su totalidad, por lo que el programa recibe los parámetros correctamente tanto en el caso de que se acceda al K2UM 2.0 con configuración que sin ella en el caso del empleo de los mismos parámetros. Por otro lado, se utilizarán los parámetros mostrados en el diagrama de la Figura 79 para el empleo de la plantilla.

Número mínimo de movimientos	Número máximo de movimientos	Tiempo Máximo en la escena
15	30	30

Figura 79. Parámetros utilizados en la plantilla por defecto

4.3. Reporte de mejoras de usuario

Para finalizar, se recoge un reporte de mejoras propuestas por los usuarios involucrados en las pruebas de funcionamiento para tener en cuenta en el juego de *Phiby´s Adventure 3D* a nivel general. Los reportes de mejoría proporcionados por estos son los enumerados a continuación:

- "Una buena opción sería tener la posibilidad de ver la cara del personaje, para que el jugador se sienta identificado con Phiby".
- "A nivel de diseño, los colores parecen muy apagados en la isla.
- "El mini mapa es muy pequeño para lo lejos que hay que estar de la cámara"
- "A veces la cámara se sale del terreno y parece que el personaje se ha salido de la isla"

Las soluciones propuestas a estos reportes se encuentran en las posibles futuras vías de trabajo, añadidas a otras propuestas en vistas al futuro del videojuego.

5. Conclusiones y posibles futuras líneas de trabajo

5.1. Conclusiones

Se procede al repaso de los objetivos aportados en la introducción del primer capítulo de este proyecto. El primer objetivo, consiste en la planificación, diseño y creación de una arquitectura general del contenido del juego. Este se considera un objetivo alcanzado, dado que, como parte de un equipo en el que se hayan acordado las mecánicas del juego, se ha generado una estructura de proyecto genérica y manejable, además de que se han integrado los parámetros entre varios proyectos.

Por otro lado, se ha conseguido el segundo objetivo relacionado con el diseño de una lógica de juego con ajuste terapéutico que consista en la instauración de una nomenclatura para un ejercicio y la fijación de parámetros ajustables para este. Además de esto, se ha creado una plantilla de valores por defecto, para la participación en el juego sin necesidad de acudir a la web.

Para finalizar, también se ha llevado a cabo de forma acertada la creación y el diseño de elementos prácticos y creativos que puedan dar cabida a futuras líneas de trabajo para la instauración de nuevas funcionalidades de juego que generen mayor interacción y reduzcan la probabilidad de fatiga por parte del jugador.

5.2. Posibles futuras líneas de trabajo

Entre futuras líneas de trabajo que pueden ser implementadas para la optimización del juego de *Phiby´s Adventure 3D* y la comunicación de este con otros elementos, en base al desarrollo de la solución propuesta en este PFG, se encuentran las siguientes:

1. Mejora de las diferentes funcionalidades objetos 3D dinámicos para una mejor experiencia de juego, como podrían ser la comunicación de estos con la Kinect 2.0 a través del *Kinect Asset*, la creación de nuevos medios de transporte para la isla en base a los existentes, creación de nuevos objetos de recolección e inventario para estos y diseño de elementos estéticos hacia una mejor estética del entorno del mundo abierto.
2. Solución para el movimiento del brazo para la detección del movimiento del cursor integrada en el *Kinect Asset* [17] que sirva para la utilización del cursor en los diferentes menús del juego.
3. Diseño de una lógica de ajuste de juego para el desarrollo de un equilibrio entre la seguridad proporcionada al usuario de que será capaz de obtener logros combinada con el riesgo de poder perder ciertos progresos mediante el sistema de *respawn*.
4. Diseño de una lógica de juego mediante la creación de una línea temporal a través del *Game Manager*, en otras palabras, creación de una historia dentro de la aventura.
5. Perfeccionamiento del sistema de control del personaje principal mediante Kinect: sustitución del sistema de ángulos mínimos en el giro del personaje por un sistema de giro más suavizado y continuo.
6. Generación de conexiones de nuevas escenas con la web a partir de la creada en la solución de este TFG, como la ya implementada de "*Chop the wood"* y construcción de un ajuste de retorno de resultados a través del *Kinect Asset* y el *Exercise Manager* integrado.

Por otro lado, las posibles futuras líneas del proyecto en base al reporte de usuarios obtenido en las pruebas de funcionamiento de este son:

1. Creación de un controlador de cámara en tercera persona a través de movimientos corporales para un dominio de esta más significativo.
2. Actualización de texturas y a una versión plana para la generación del estilo *low poly* y desarrollo de sistemas renderizado de imagen en el juego.
3. Creación de mapa de juego e implementación de apertura y cierre de este a través de movimientos corporales con la Kinect 2.0. Además, creación de un sistema de zoom del mini mapa mediante movimientos corporales.
4. Actualización de controlador de *colliders* de la cámara en tercera persona para que esta no se salga del *terrain* de la isla en el giro.

Referencias

[1] Kinect for Developers (2013). Características Kinect 2 [Online]. Disponible en: http://www.kinectfordevelopers.com/es/2014/01/28/caracteristicas-kinect-2/

[2] Blender (2020). Blender [Online]. Disponible en: https://www.blender.org/

[3] C. Luaces, “Diseño e implementación de un entorno virtual de ejercicios físicos, basados en captura de movimiento”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2018.

[4] Unity Technologies (2020) [Online]. Disponible en: https://unity.com/products

[5] A. Aguilar, “Implementación de medidas de seguridad en plataforma médica para entorno de videojuegos terapéuticos”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2019.

[6] NeuronUP (2020). Entrenamiento cerebral personalizado [Online]. Disponible en: https://www.neuronup.com/es/

[7] Red Menni de Daño Cerebral (2015). Aplicación de las nuevas tecnologías para la rehabilitación de las personas con daño cerebral [Online]. Disponible en: https://xn--daocerebral-2db.es/aplicacion-de-las-nuevas-tecnologias-para-la-rehabilitacion-de-las-personas-con-dano-cerebral/

[8] H/P/Cosmos (2020). Robowalk expander B 150/50 [Online]. Disponible en: https://www.hpcosmos.com/en/robowalk-expander-b-15050

[9] Umana (2018). Tecnología vinCi (Realidad Virtual Rehabilitación) [Online]. Disponible en: http://www.umana.es/vinci-realidad-virtual-rehabilitacion/

[10] Wikipedia (2020). Unity (motor de videojuego) [Online]. Disponible en: https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

[11] Unity Technologies (2016). Documentación [Online]. Disponible en: https://docs.unity3d.com/es/530/Manual/UICanvas.html

[12] Digital Learning SL (2016). Academia Android [Online]. Disponible en: https://academiaandroid.com/scripts-lenguajes-programacion-unity/

[13] Wikipedia (2019). Kinect [Online]. Disponible en: https://es.wikipedia.org/wiki/Kinect

[14] I. Gómez-Martinho, “Desarrollo e implementación de middleware entre Blender, Kinect y otros dispositivos”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2016.

[15] M. Jiménez, “Plataforma médica para el entorno de videojuego terapéutico Blexer”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2017.

[16] D. Iglesias, "Middleware K2UM 2.0", Informe de Prácticas, Universidad Politécnica de Madrid, España, 2019.

[17] G. Cilleruelo, “Solución de juegos *First Person Shooter* (FPS) orientado a personas con movilidad reducida”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2019.

[18] L. Molero, “Diseño e implementación del entorno para el juego serio terapéutico *Phiby’s Adventures V2*”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2019.

[19.] C. Esteban, “Diseño e implementación del entorno de videojuego serio de rehabilitación del proyecto Blexer”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2017.

[20] M.A. Gil, “Integración y calibración de movimientos captados mediante la cámara Kinect para el juego serio terapéutico *Phiby’s Adventures V2*”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2019.

[21] P. López, “Diseño e implementación de un videojuego de ejercicio físico para personas con discapacidad”, Proyecto Fin de Grado, Universidad Politécnica de Madrid, España, 2019.

[22] Y. Chou’s, “Octalysis Game-Techniques”, Actionable Gamification: Beyond Points, Badges, and Leaderboards, 2015.

[23] Unity Asset Store (2020). Assets[Online]. Disponible en: https://assetstore.unity.com/

[24] M. Eckert, J.A. García, H. Quin, L. Molero, M.A. Gil, J.D. Guerrero, C. Luaces. “Documento de diseño para: *Phiby´s Adventure 3D*”. 2020

Anexo I: Funcionalidades de algunas de las componentes del Kinect Asset

- ***Launcher.cs***: Se utiliza para la definición de variables públicas. En ella se definen variables a las que el resto de clases del *Kinect Asset* accederán de forma compartida.
- ***Rotation.cs***: Clase que permite la rotación de los GameObjects asociados en ella, en función de la rotación recibida a través del middleware y almacenadas en la cola de mensajes pendientes de procesar, *messageQueue*. Además, y en función de las activaciones de articulaciones que se realicen en la clase *Launcher*, se genera, con cada modificación de este tipo un mensaje con las activaciones realizadas que se enviara al middleware. La clase genera un hilo secundario para la recepción que se ejecutara en segundo plano, y que permanecerá a la espera de que se encolen nuevos mensajes en la cola de mensajes pendientes de procesar definida en la clase *Launcher*. Actualización de Daniel Iglesias para el envío de la solicitud de configuración y la recepción de los datos de configuración de Blexer-Med.
- ***UDPReceive.cs***: Clase que permite la recepción de mensajes a través de un canal UDP. Los mensajes consistirán en cadenas de texto que la clase procesará y añadirá a la cola de mensajes pendientes de procesar definida en la clase *Launcher*. En dicha cola se almacenarán los mensajes con las rotaciones de las articulaciones activas definidas en la clase *Launcher*. La clase genera un hilo secundario para la recepción que se ejecutara en segundo plano, y que permanecerá a la espera de recibir nuevos mensajes con información sobre las rotaciones de cada articulación.
- ***UDPSend.cs***: Clase que permite el envío de mensajes a través de un canal UDP. Los mensajes consistirán en cadenas de texto que la clase procesara y enviara desde el cliente UDP.
- ***HeadersUDP.cs***: Clase con las cabeceras utilizadas en el envío y recepción de mensajes UDP.
- ***SettingManager.cs***: Clase que gestiona la solicitud, recepción y deserialización de la configuración.

Anexo II: Asignación del personaje Phiby al *KinectAsset* (redactado por Miguel Ángel Gil)

Para el desarrollo de este trabajo de fin de grado, ha sido necesario utilizar el *asset* desarrollado '*KinectAsset'*, el cual es el encargado de copiar los movimientos del jugador y, a través de la cámara *Kinect*, traducirlos al entorno de juego y aplicarlos al personaje de la manera deseada.

En primer lugar, el personaje 'Phiby V1' ya ha sido introducido en el mundo del juego y, por tanto, puesto en escena. A su vez, el asset '*KinectAsset'* también ha sido introducido en la escena. Se ha creado un *GameObject* inicial, vacío, donde se han agrupado tanto nuestro personaje como el *asset* para recoger y copiar los movimientos al mismo.

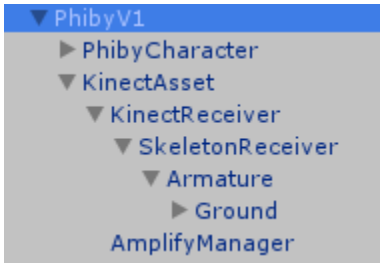


Figura 1: Agrupación del *asset* y del personaje *Phiby* dentro de un *GameObject* vacío

Se ha utilizado esta metodología de trabajo y agrupación para poder manejar, de manera más sencilla, todo el conjunto del personaje dentro de la escena y del juego. Además, de esta forma se ha generado el *prefab* que se ha exportado a modo de paquete externo (*unityPackage)*, para que pueda ser utilizado por cualquier integrante del equipo y realizar pruebas o ejecuciones en el juego sin depender del avance de otro compañero.

A continuación, se describen cada uno de los elementos que contiene este *GameObject* llamado '*PhibyV1'*. '*PhibyCharacter'* contiene todas las animaciones de nuestro personaje (reposo, andar y correr), así como las texturas y materiales de este y su esqueleto. En él, el elemento '*Armature'* referencia al esqueleto formado por los huesos que se importaron desde Blender, los cuales son asociados al asset para asociar las extremidades encargadas de copiar los movimientos del jugador.

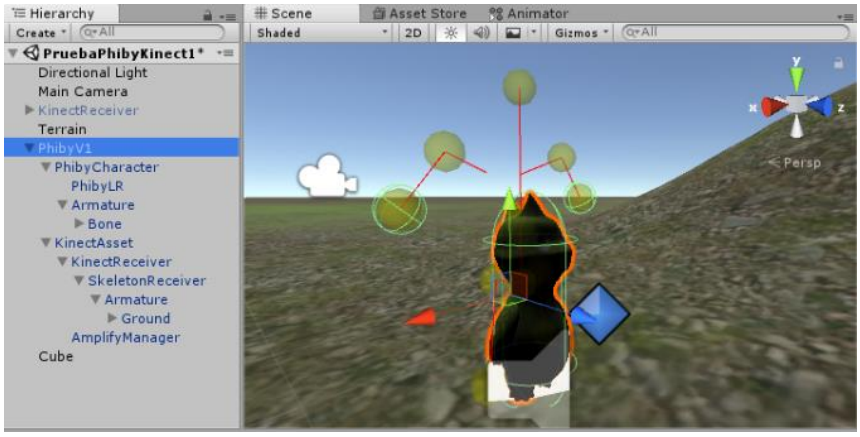


Figura 2: Escena del juego donde se muestra desglosado el *GameObject PhibyV1* junto a cada elemento de este

Asociado a '*Armature*', se han añadido una serie de componentes específicos necesarios para permitir el desplazamiento del personaje, que este quede afectado por la gravedad del juego, que sea capaz de responder ante las variedades del entorno y del suelo y que choque o no avance cuando tenga obstáculos en su camino.

Figura 3: Pestaña de juego junto a los componentes asociados al '*Armature*' del personaje

La componente *Transform* corresponde a la posición dentro del mapa de juego donde se encuentra el personaje, junto con su rotación y escala.

Las más importantes son *RigidBody* y *Capsule Collider*. La primera es necesaria para poder indicar que este elemento del juego, este *GameObject*, tiene que obedecer las reglas de la física dentro de ese mundo. Por lo tanto, se le asocia un valor de masa inicial, y que el mismo reciba la gravedad marcando el *checkbox* como válido. Pero dentro de este componente, es muy importante señalar la pestaña desplegable de *Constrains*. Esta es la encargada de indicar sobre que ejes, de traslación y rotación, va a quedar afectado nuestro personaje.

Como se puede ver en la figura anterior, se han marcado las casillas correspondientes de '*Freeze Position*' y '*Freeze Rotation*'. Dentro de '*Freeze Position*', se ha marcado únicamente la X, dado que las coordenadas del '*Armature*' de nuestro personaje están cambiadas. Por lo tanto, si se "congela" la posición en X quiere decir que cualquier obstáculo que haya en ese eje bloqueará el paso del personaje o bien lo hará chocar con el elemento en cuestión. Lo mismo ocurre con la componente en el eje Z.

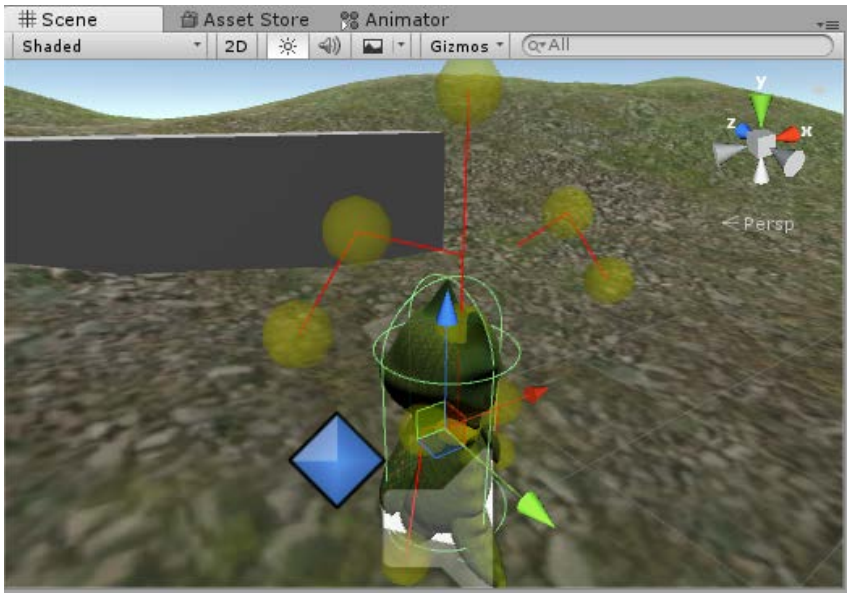


Figura 4: Pestaña de la escena del juego donde se muestran los ejes XYZ donde se observa que si dejando libre la coordenada Y, el personaje chocará con los elementos del plano y será capaz de subir y bajar pendientes debido a que la componente en el eje Y está liberada. Con Freeze Rotation se evita que el personaje rote en el juego (XYZ)

De este componente, lo comentado anteriormente es lo más destacable. El siguiente elemento que posee este *GameObject* es un *Capsule Collider*. Este elemento permite que el personaje quede afectado por cualquier objeto dentro del entorno de juego, lo que permite que al estar cerca de otros elementos o bien de ciertas zonas de juego actúe como un *Trigger* lanzando el *script* en cuestión o cambiando de escena, entre otras posibilidades.

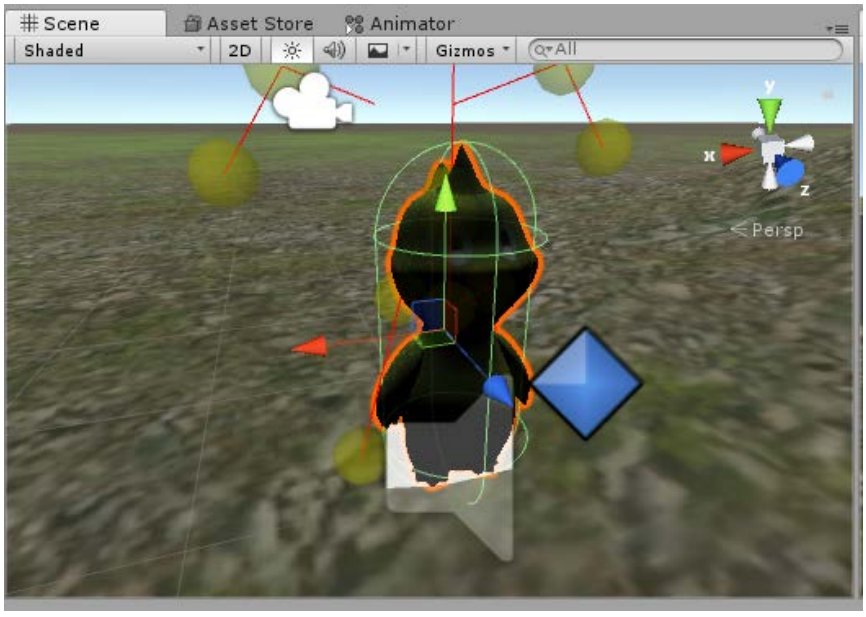


Figura 5: Pestaña de la escena del juego donde se muestra el *Capsule Collider* (componente ovalada que rodea al personaje) la cuál debe ser lo más similar posible al contorno del personaje para que cualquier choque o acción se asemeje con la realidad

Por otro lado, se asocia también al *GameObject PhibyV1* el *KinectAsset*, mediante el cual los movimientos que vayan a ser realizados por el jugador se van a copiar al personaje. Esto se indica asociando cada hueso del personaje al *slot* correspondiente dentro del *script* principal de este *asset*.

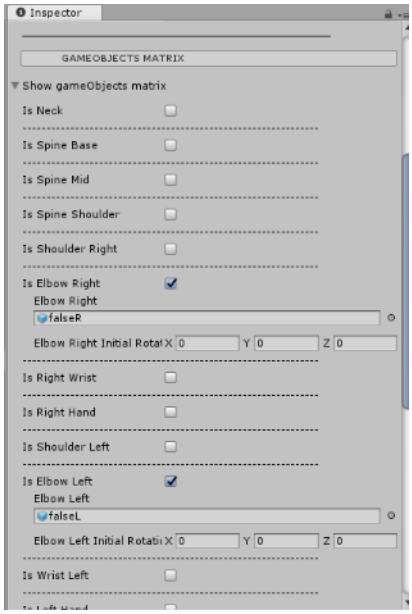


Figura 6: Script asociado al *KinectReceiver* (dentro del *KinectAsset*) donde se asocian los huesos recogidos

En este caso, para detectar que el movimiento del jugador se realice girando el tronco para rotar, e inclinando el tronco hacia delante o hacia atrás para avanzar o parar, los huesos se han asociado a los hombros del asset, (*Elbow Left*/*Right*). Finalmente, para evitar que el personaje se doblara en cada inclinación o rotación se ha establecido unos *GameObjects* vacíos, a la altura del tronco del personaje *Phiby* que son los encargados de asociar las inclinaciones del jugador y traducirlas al desplazamiento y movimiento del personaje sobre el juego.

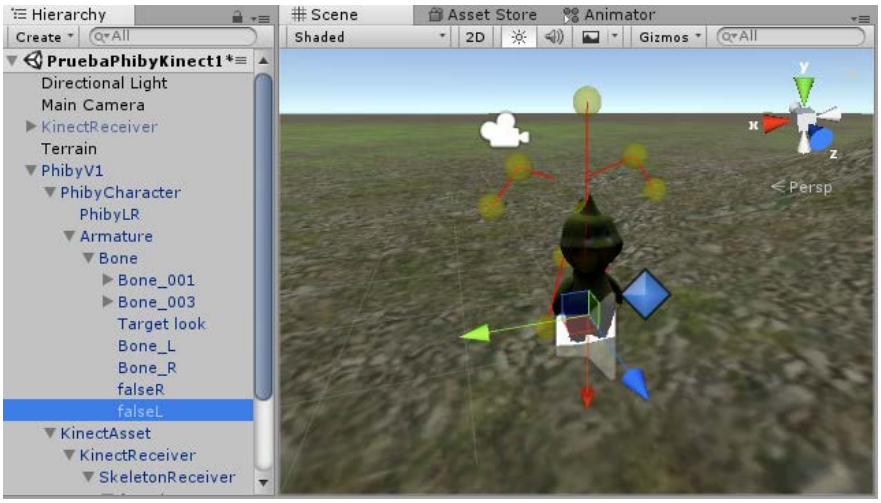
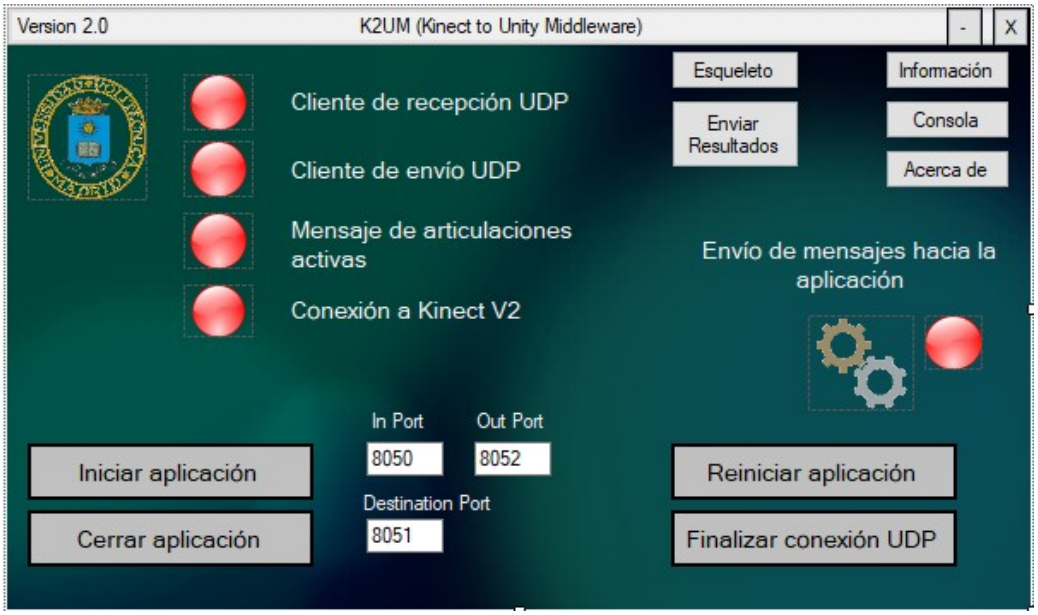


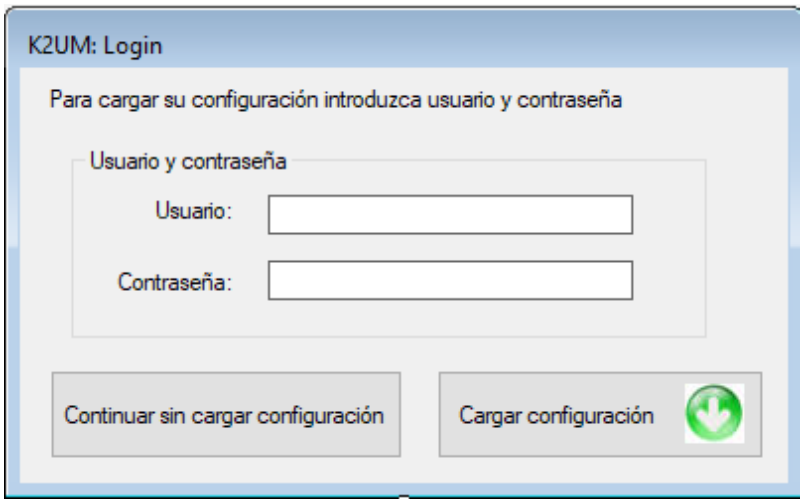
Figura 7: Asociación de los GameObjects vacíos (*falseL*, *falseR*) a la altura central del tronco del personaje, junto con los huesos del asset (Elbow Left/ Right)

Anexo III: Resumen del funcionamiento del K2UM 2.0 – Daniel Iglesias, abril 2019

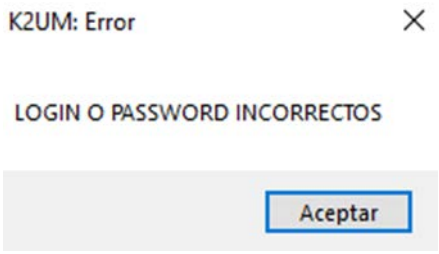


Inicio de la aplicación

Al arrancar la aplicación se muestra la ventana que pide el usuario y contraseña.



Hay dos opciones: continuar sin descargar ningún fichero de configuración o introducir un nombre de usuario y contraseña. Si este usuario no está registrado en la web o su contraseña no es correcta saltará un aviso.

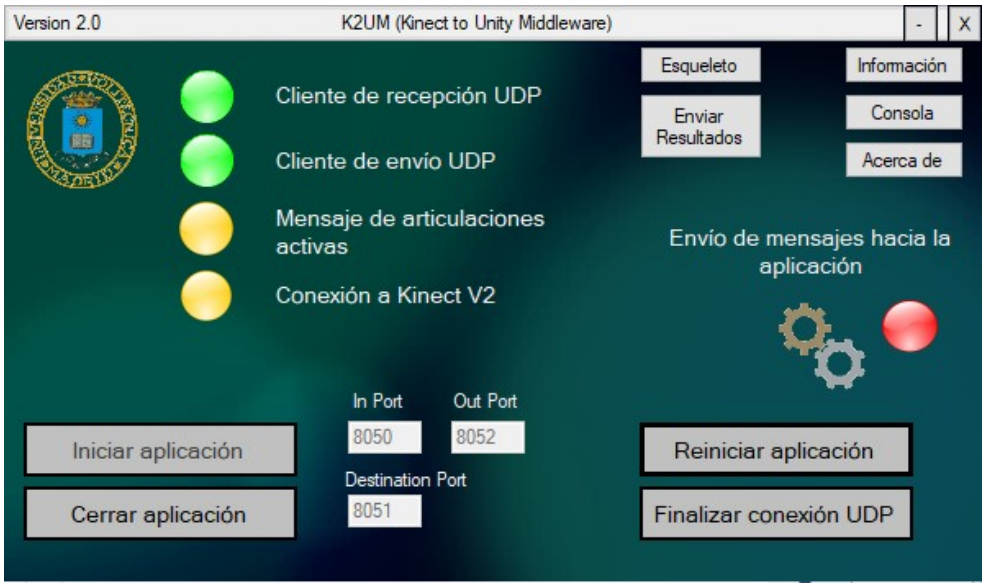


Esta parte del código se desarrolla en la clase **LoginForm.cs** la cual contiene las variables y métodos que se usan para la comunicación con la página web.

Al pulsar el botón *Cargar Configuración* se ejecuta el método **cargarConfiguración()** el cual envía un mensaje de solicitud de datos a la web mediante el protocolo HTTP. Esta parte de del código se ha copiado tal cual del middleware Chiro y se ha adaptado al actual.

La configuración que el middleware recoge de la web se guarda en un fichero de texto de nombre el login del usuario seguido de la extensión .txt. Este fichero se guarda en la carpeta donde esté instalado el programa en la subcarpeta **"\k2umfiles\users-settings**".

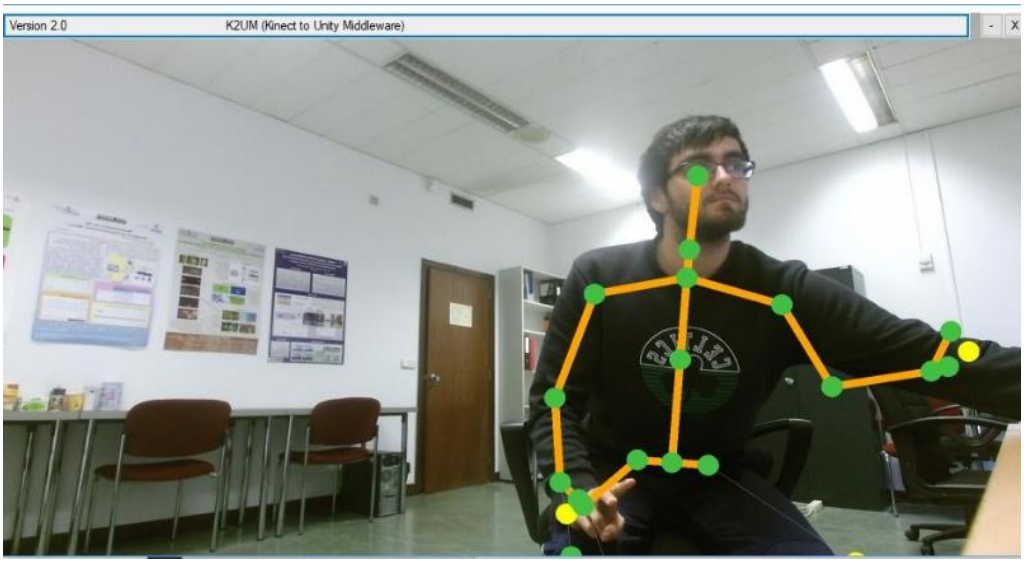
Al pasar el proceso de autenticación se abre la ventana principal gestionada por el formulario diseñado en **Form1.cs**. Al pulsar el botón *Iniciar aplicación* comienza la comunicación entre el middleware y la Kinect.



Ventana Kinect Skeleton

Una vez iniciada la conexión se pueden usar el resto de funciones del middleware como la información de los mensajes enviados o la ventana de depuración para el seguimiento de la aplicación.

Al pulsar el botón *Esqueleto* se abre la ventana independiente que muestra la imagen de la cámara de Kinect y dibuja si detecta algún cuerpo el esqueleto del mismo.



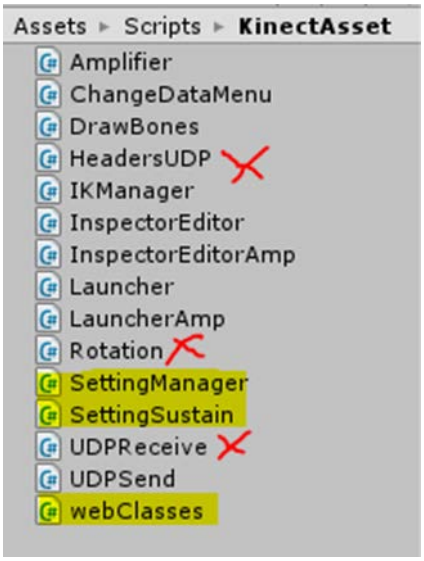
El seguimiento del código utilizado comienza en la clase **Form1.cs.** Al pulsar el botón *Esqueleto* se crea un hilo secundario llamado **skeletonThread** el cual activa la ventana de la cámara. El resto del proceso se realiza en la clase **Form2.cs**.

En ella se toman los datos que manda la Kinect (30 frames por segundo) y se diferencian si son datos de imagen (**ColorFrame)** o datos de los esqueletos

(**BodyFrame**). Ambos datos se combinan para formar la imagen resultante. Kinect puede captar hasta 6 esqueletos distintos al mismo tiempo.

Kinect Asset

Una vez iniciada la conexión con Kinect se puede establecer conexión con Unity a través del código incluido en el asset. Al iniciar una escena que contenga el objeto kinectReceiver del asset de Kinect, que puede ser por ejemplo el menú principal, se inicializan todas las variables para la conexión UDP con el middleware y se envía la solicitud de datos de configuración al mismo. Este proceso involucra los scripts **udpSend.cs y settingManager.cs**.



En amarillo, scripts nuevos añadidos. En rojo scripts existentes modificados.

Para que este proceso sea posible se han implementado dos nuevos tipos de mensajes de control, mensaje de solicitud de configuración (*Setting Request* – SR) y mensaje de configuración enviada desde el middleware (*Setting Sent* - SS).

Desde la clase **SettingManager.cs** se solicita el archivo de configuración tras cargar la escena. El middleware recibe la solicitud de configuración (SR), la cual contiene información del juego cuyos datos solicita y tras esto deserializa la información correspondiente al juego solicitado, contenida en el fichero de configuración descargado. La información del juego se serializa de nuevo en formato JSON y se envía a Unity (SS) donde la clase **UDPReceive.cs** se encarga de recibir el mensaje, comprobar qué tipo de mensaje es y colocarlo en la cola de mensajes recibidos. La clase **Rotation.cs** desencola el mensaje y extrae la cadena de texto en formato JSON. La información del JSON se deserializa en la clase **SettingManager.cs** cuyo contenido es la configuración de todos los ejercicios del juego y los guarda en un diccionario de ejercicios. Este diccionario está definido en la clase **SettingSustain.cs** que se asignará como componente a un gameObject "*settingObject*" que no se destruirá al cambiar de escena y permitirá recoger la configuración para cada ejercicio.

También se ha desarrollado un ejemplo de script para extraer la configuración y al terminar el ejercicio mandar los resultados, como ayuda para los desarrolladores de los juegos. Este script se llama **exerciseManager.cs.**

Cuando el usuario decida puede enviar los resultados de los ejercicios obtenidos hasta el momento usando el botón “Enviar Resultados” del middleware. Los resultados que se van llegando se guardan en un fichero de texto de nombre el login del usuario seguido de la extensión .txt y almacenado en el directorio del middleware dentro de **“\k2umfiles\users-settings**”.

El componente **SettingManager.cs** debe estar presente en el objeto *kinectReceiver* en la escena en la que se decida solicitar el archivo de configuración, por ejemplo, el menú del juego.

Todas las clases que estructuran los datos de configuración y resultados que se envían a la web se guardan en el fichero **webClasses.cs**.

Anexo IV. Plantilla de valores por defecto

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class gameParametersTemplate : MonoBehaviour
{
/* gameParametersTemplate V 1.0 - Designer: Juan Alberto García - Update:
20/01/20 Class that save the default Parameters of the game in case the
configuration parameters coundn't be used.
*/
    public bool isDontDestroy = false;
    /* Climb Apple Tree - idExercise = 6009 */

    public string num_mov_min = "20";
    public string num_mov_max = "35";
    public string time_max = "10";
    public string Ex1param4 = "0";

    /* Climb Apple Tree */

    /* Chop Wood - idExercise =  */

    public string Ex2param1 = "";
    public string Ex2param2 = "";
    public string Ex2param3 = "";
    public string Ex2param4 = "";

    /* Chop Wood */

    /* Boat - idExercise =  */

    public string Ex3param1 = "";
    public string Ex3param2 = "";
    public string Ex3param3 = "";
    public string Ex3param4 = "";

    /* Boat */

    /* Ski - idExercise =  */

    public string Ex4param1 = "";
    public string Ex4param2 = "";
    public string Ex4param3 = "";
    public string Ex4param4 = "";

    /* Ski */
    public void Awake()
    {
        if (!isDontDestroy)
        {
            DontDestroyOnLoad(this);
        }
    }
}
```

Anexo V. Código fuente

Debido al hecho que las herramientas creadas son muy valiosas para la continuación de la línea de investigación de videojuegos para rehabilitación, llevado a cabo en el grupo de investigación GAMMA en el que se realizó este proyecto, no se va a publicar el código fuente en este libro. Sin embargo, estará a disposición para las personas interesadas en contribuir a una ampliación o mejora del software. En este caso, se tendrá que poner en contacto con la tutora Martina Eckert.

Sin embargo, se proporciona un CD con el software fuente y dos ejecutables a los profesores que constituyen el tribunal. En este CD se encuentran:

PhibysAdventures3D_Teclado.exe

Ejecutable para Windows 10. Se puede mover el personaje principal por la isla con las teclas del ordenador (flechas adelante, izquierda y derecha). Sin embargo, no se puede entrar en las escenas de ejercicio, porque solo funcionan con la Kinect.

PhibysAdventures3D_Kinect.exe

K2UM3.0.zip

Ejecutable para Windows 10. Se puede jugar con una Kinect 2.0 conectada, después de ejecutar el middleware K2UM 3.0 que también se adjunta.

PhibysAdventures3D_UnityProyect.zip

Proyecto de Unity con el código fuente, creado bajo la versión Unity LTS 2018.4

Anexo VI. Presupuesto

En este anexo se elabora un presupuesto de los costes generales del proyecto, tanto a nivel económico del material utilizado, como a nivel de horas de trabajo dedicadas al desarrollo del proyecto y a la redacción de la memoria.

Materiales:

- Equipo 1...……...400 €
  - ✓ Intel® Core 2 6400 a 2.13 GHz
  - ✓ 4 GB RAM DDR3
  - ✓ 500 GB Disco Duro 5400 rpm
  - ✓ Intel Graphics

- Equipo 2……..1000 €
  - ✓ Intel® Core i7-4790 a 3.6 GHz
  - ✓ 16 GB RAM DDR3
  - ✓ 1000 GB Disco Duro 5400 rpm
  - ✓ 128 GB SSD
  - ✓ 4 GB ATI Radeon

- Microsoft Kinect 2.0…….....150 €
- Presupuesto TOTAL ……..1550 €

Horas de trabajo:

- 6 meses de del desarrollo del aporte de diferentes elementos al juego
  - ✓ 5 horas semanales ……120 horas totales

- 4 meses de desarrollo de la solución propuesta
  - ✓ 25 horas semanales ……..400 horas totales

- 2 meses realización pruebas y redacción informe
  - ✓ 20 horas semanales ……160 horas totales