ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Diseño e implementación de un videojuego de ejercicio físico para personas con discapacidad

AUTOR: Pablo López Miedes

TITULACIÓN: Grado en Ingeniería de Imagen y Sonido

TUTOR: Martina Eckert

DEPARTAMENTO: Teoría de la Señal y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Javier Martín Rueda

TUTOR: Martina Eckert

SECRETARIO: Enrique Rendón Angulo

Fecha de lectura: 5 de marzo de 2019

Calificación:

El Secretario,

Agradecimientos

A mis padres, por el apoyo y la confianza depositadas en mí al iniciar una carrera en la gran ciudad. Sin vosotros no hubiese llegado a donde me hallo ahora.

A mi novia Irene, por ser mi faro, por la paciencia y el respaldo. Por aguantar estoicamente a mi lado en esta aventura. Gracias.

A todas las personas que he conocido en Madrid, que han recorrido parte del camino conmigo y que me han demostrado que el hogar no conoce de lugares: Adri y su familia, Alberto, Gabri, Laura, Cris, Yago, Angélica, Borja, Guille, Rocío.

A los profesores gracias a los cuales tengo una gran mochila llena de conocimientos técnicos. En especial a mi tutora Martina por permitirme llevar a cabo este proyecto.

A todas las personas que en algún momento me han ayudado en esta etapa.

Resumen

El objetivo de este proyecto es el desarrollo de un videojuego controlado mediante movimientos corporales. Para ello se ha utilizado el hardware de captación de movimientos Kinect v2, el middleware K2UM y el software de creación de videojuegos Unity 3D.

Este trabajo se enmarca dentro del grupo de investigación GAMMA (Grupo de de Aplicaciones MultiMedia y Acústica), con sede en el CITSEM, y su proyecto BLEXER (Blender Exergames) que tiene la finalidad de crear una serie de aplicaciones médicas para ayudar en la recuperación física de personas con algún grado de discapacidad. El objetivo es por tanto monitorizar el desarrollo de dicha recuperación a través de ejercicios guiados dentro de un videojuego.

Se parte del videojuego *Buscando a Totoro*, desarrollado en Blender para la asignatura de Síntesis y Animación de Imágenes, por lo que se hace un gran hincapié en los problemas de compatibilidad que tienen ambos softwares y los formatos de interoperabilidad que permiten exportar e importar elementos entre ellos. También se detalla la integración de las herramientas Kinect v2 y K2UM, necesarias para implementar la interfaz natural de usuario que permite controlar el videojuego a partir de la captación de movimientos.

Así, el resultado es un juego funcional que permite al usuario ejercitar movimientos corporales tales como la inclinación del tronco o el alzamiento de brazos dentro de un entorno virtual alejado de la rehabilitación clásica.

Abstract

The main objective of this project is the development of a video game controlled by body movements. For this purpose, Kinect v2 movement capture hardware, K2UM middleware and Unity 3D video game engine have been used.

This work is part of the research group GAMMA (Group of MultiMedia Applications and Acoustics), located in CITSEM, and its BLEXER project (Blender Exergames), that aims at creating a series of medical applications to help in the physical recovery of people with some degree of disability. Therefore, the objective is to monitor the development of this recovery through guided exercises within a video game.

The base of this project is the videogame *Buscando a Totoro*, developed in Blender for the course in Image Animation and Synthesis. For that reason, a great emphasis is placed in explaining the compatibility issues that arise between both software and the interoperability formats that allow exporting and importing elements between them. It also details the integration of the Kinect v2 and K2UM tools, necessary to implement the natural user interface that allows to control the videogame through movement capture.

Thus, the result is a functional game that allows the user to exercise body movements such as trunk tilt or arm raising within a virtual environment away from conventional rehabilitation.

Índice de Contenido

Índice de figuras

Lista de acrónimos

BaT: Buscando a Totoro.

BGE: *Blender Game Engine.*

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

GAMMA: Grupo de Aplicaciones Multimedia y Acústica.

K2UM: *Kinect 2 (to) Unity Middleware*.

LIDAR: *Light Detection and Ranging*.

SAI: Síntesis y Animación de Imágenes. Asignatura de tercer curso de la titulación de Grado en Ingeniería de Sonido e Imagen.

1. Introducción

Desde hace unos años hay una corriente de pensamiento que ha tomado fuerza en el ámbito empresarial y se ha ido extendiendo hacia otros ámbitos. Dicha corriente parte de la idea de que la gente disfruta jugando a videojuegos y propone transmitir esa sensación a otras actividades, es decir, *"gamificar"* diferentes entornos. "Ludificación" o "gamificación" (del inglés *Gamification*) es el proceso de extraer los elementos adictivos y divertidos de los juegos y aplicarlos a procesos productivos del mundo real.

Los primeros indicios de la incorporación de los videojuegos a la rehabilitación física datan de los años 80 [1], sin embargo, la reciente irrupción en el mercado de software que permite implementar interfaces naturales de usuario y el hardware correspondiente a un precio asequible abre infinidad de posibilidades. Este es un campo todavía con mucho por explorar y con un gran potencial. Así lo demuestran numerosos estudios como el realizado por la universidad de Oxford para su revista *Physical Therapy* [2] en el que se obtuvieron resultados satisfactorios en el tratamiento de un adolescente con parálisis cerebral mediante el entorno virtual de Wii Sports o el realizado por la universidad de Essex que consistía en evaluar la eficacia de utilizar la Nintendo Wii para mejorar el equilibrio y la movilidad posteriores a una caída [3], por nombrar algunos.

En líneas generales los pacientes agradecen salir de la monotonía de sus ejercicios habituales y se sienten motivados por este tipo de ejercicios. Sin embargo, es necesario tener en cuenta el contexto en el que se realizan y deben ser siempre supervisados y monitorizados por personal médico para evitar los riesgos de incurrir en nuevas lesiones al utilizarlos.

El grupo GAMMA [4] (Grupo de Aplicaciones Multimedia y Acústica), enmarcado en el CITSEM [5], viene trabajando en el desarrollo de proyectos que aúnan los videojuegos y la captura de movimiento en 3 dimensiones para pacientes con algún tipo de disfunción motora. El proyecto Blexer (conjunción de las palabras Blender y *Exergames*) es uno de ellos, y pretende crear un entorno con diferentes videojuegos que, por un lado, faciliten la abstracción de un proceso que puede resultar desmotivante y aburrido para los pacientes y por otro, permita la monitorización de su desarrollo para los especialistas médicos.

En una primera fase del proyecto se pretendía desarrollar aplicaciones con el software de modelado 3D Blender. Ignacio Gómez-Martinho desarrolló el middleware llamado "Chiro" [6] que permite la comunicación entre la cámara Kinect X360 de Microsoft y un juego creado en Blender. El middleware transmite los movimientos del usuario, captados por la cámara, al software Blender, que a su vez traduce estos movimientos en acciones dentro del juego. En este entornó se creó el primer juego completo llamado "Phiby's Adventures", en el cual el usuario lleva al personaje principal por un entorno a explorar. En este entorno tiene que realizar una serie de ejercicios para ir creciendo fuerte y sano. Los ejercicios y sus correspondientes ejercicios físicos consisten en cortar madera (mover el brazo arriba y abajo), escalar (mover ambos brazos alternativamente, simulando una

escalada), bucear por un lago y comer plánctones (controlado a través de los movimientos del tronco) y remar (simular el remo).

Debido a que Blender tiene un motor de juego muy reducido, se decidió migrar el software de desarrollo a Unity 3D, un motor de juegos con mayor versatilidad y potencia. Además, se optó por usar la versión más reciente de Kinect, la v2 de la Xbox One, que permite mayor precisión y más puntos de detección. Para ello hubo que crear un nuevo middleware, lo cual hizo César Luaces en su Proyecto fin de Grado [7]. El nuevo middleware se llama K2UM (Kinect to Unity Middleware). En la actualidad se está realizando una nueva versión de "Phiby's Adventures" [8] para este entorno además de crear otros juegos de diferente género para el entorno Blexer.

Por otra parte, el sistema Blexer incluye una plataforma web llamada "Blexer-med" realizada por Mónica Jiménez [9][10], que permite a los especialistas médicos el seguimiento y la gestión del progreso de sus pacientes, así como la configuración de la dificultad de los videojuegos según el grado de habilidad del paciente.

En la actualidad, este proyecto se sigue desarrollando incluyendo nuevos juegos y mejoras a los ya existentes. Aquí es donde entra en escena este trabajo de fin de grado, ya que pretende incluir un juego más al entorno. El videojuego que se va a desarrollar, *El desafío de Fruitman*, proviene de un proyecto anterior realizado para la asignatura de SAI. Su predecesor, Buscando a Totoro, es un juego creado en Blender por Cristina Corporales, Angélica Suarez y Pablo López (autor de este documento). En él, Fruitman, un carismático personaje formado por frutas, tiene que rescatar a su inestimable amigo Totoro, que ha sido secuestrado por los malvados demonios (Yokais). Consta de dos niveles y en ambos el personaje se encuentra en el interior de un laberinto. En el primer nivel Fruitman tiene que adentrarse en él para encontrar a Totoro. Al encontrarlo se da paso al siguiente nivel, en el que ambos deben salir del laberinto a contrarreloj. Los Yokais son los enemigos e intentan evitar que Fruitman consiga su objetivo, así que lo siguen para atacarle y restarle vida.

Este juego tiene varias ventajas para poder ser utilizado para ejercicios terapéuticos:

- Tiene lugar en un ambiente amigable e incorpora a un personaje muy conocido por los jóvenes (Totoro), con lo cual anima mucho a ser jugado por ellos.
- El jugador se puede identificar fácilmente con el personaje de Fruitman al ser el protagonista que debe encontrar a Totoro y defenderse contra los malos.
- Los movimientos que tiene que hacer Fruitman, andar y pegar, son casi directamente transferibles desde los movimientos del usuario. El usuario debe estar fijo delante de la cámara, además, el juego debe ser apto para personas en silla de rueda, con lo cual la ambulación debe ser simulada por el juego. La dirección de los movimientos se puede simular mediante giros del cuerpo. Los ataques se pueden simular mediante movimientos de los brazos.

1.1. Objetivos del proyecto

El objetivo principal de este proyecto es reutilizar el videojuego "Buscando a Totoro" y transformarlo al entorno de Unity 3D, integrando el control por una interfaz natural de usuario, en este caso la cámara Kinect v2, para que forme parte del proyecto Blexer.

Este objetivo se puede desgranar en tres grandes bloques de trabajo.

- Exportación de *Buscando a Totoro* de Blender e importación a Unity.
    - Definición de los elementos a exportar.
    - Modificación de dichos elementos para prevenir problemas en la exportación, p.ej. alinear el origen de coordenadas del objeto con su malla.
    - Exportación en formato *.fbx*.
    - Importación en Unity 3D.
- Generación de la lógica asociada, basada en scripts C#.
    - Creación de menús y de las diferentes escenas del videojuego.
    - Incorporación de la banda sonora.
    - Generación constante de enemigos aleatoriamente desde diferentes puntos.
    - Creación de los scripts de movimiento de Fruitman y de seguimiento de Totoro y de los Yokais enemigos.
    - Movimiento y seguimiento de la cámara.
    - Gestión de vida y ataques de Fruitman y los enemigos.
- Integración con la Kinect v2 para la captación de movimiento.
    - Instalación del middleware y los drivers necesarios.
    - Modificación de los scripts asociados al módulo receptor del middleware para adaptarlo a las necesidades del proyecto, p.ej. girar más de 180 grados.

2. Estado del Arte

2.1. El proyecto Blexer

El proyecto Blexer tiene como objetivo la rehabilitación de personas con discapacidad física a través de los videojuegos. Su nombre proviene del acrónimo formado al unir las palabras Blender y Exergames, por basarse en la creación de los videojuegos con el software de animación Blender. Exergames a su vez es la combinación de las palabras inglesas *exercise* y *game* y generalmente utilizado en la literatura.

Inicialmente se utilizó el hardware de captura de movimientos Kinect V1 para desarrollar una serie de minijuegos en Blender aptos para controlar mediante interfaz natural de usuario. Al no haber herramientas para la comunicación entre Blender y Kinect, se decidió desarrollar el middleware Chiro, que permite a Blender recibir datos de Kinect y otros dispositivos.

Una vez asentadas las bases, comenzó una fase de mejora y expansión. Para ello se integraron los cuatro minijuegos en una historia envolvente en la que el usuario pudiese experimentar un hilo conductor y la abstracción de estar realizando ejercicios de rehabilitación. Phibys Adventure [11] es el mundo ideado para ello. El primer juego completo del proyecto, en el cual Phiby es un pequeño anfibio que deberá ir completando los ejercicios para crecer y avanzar en el juego.

Por otra parte, se desarrolló una plataforma web que permite a los especialistas médicos monitorizar y personalizar las sesiones de juegos de los pacientes. Dicha plataforma consta de una página web y una base de datos JDBC donde almacenar los resultados de cada jugador. La solución se llamó Blexer-med y es un módulo transversal a todo el proyecto Blexer, ya que puede ser utilizada tanto con Blender como con Unity y además con cualquier tipo de juego. El entorno del sistema completo queda reflejado en la figura 1 [12].

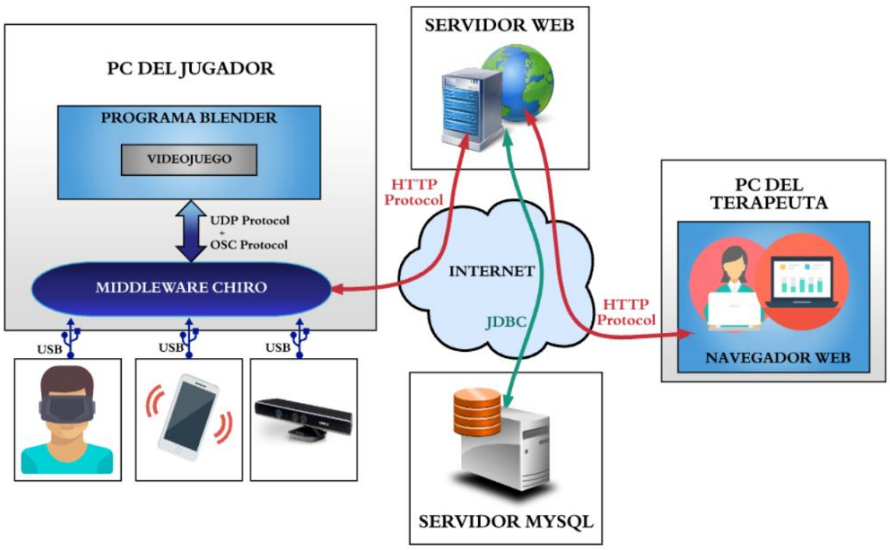

Figura 1. Diagrama de bloques del proyecto Blexer.

Posteriormente se decide migrar el entorno de desarrollo de videojuegos a Unity 3D debido a las limitaciones que tiene Blender como motor de juegos al ser principalmente un software de modelado 3D. Aprovechando el cambio de software y, puesto que había que crear de nuevo un middleware que permitiese la comunicación entre el hardware y el software, se decidió introducir la Kinect V2 en el proyecto Blexer, que incluía una serie de mejoras sustanciales frente a su predecesora. Dicho middleware, bautizado como K2UM *(Kinect to Unity Middleware)*, lo desarrolla César Luaces como parte de su TFG y permite la comunicación entre Kinect V2 y software de desarrollo de videojuegos.

2.2. El middleware K2UM

Debido a la necesidad de migrar a Unity, se plantearon varias alternativas para la interconexión entre el motor de juegos y la cámara Kinect v2. Podía implementarse directamente a través de scripts en el propio desarrollo del videojuego ya que Unity proporciona las herramientas necesarias para ello. Sin embargo, se decidió crear una herramienta complementaria que hiciese de intermediario entre ambas partes, el middleware K2UM (Kinect to Unity Middleware). De esta manera se facilitaba la creación de nuevos videojuegos dentro de Blexer a la par que abría las puertas a una futura migración a otro entorno de software distinto porque la comunicación con la Kinect sería independiente.

La cámara Kinect capta imágenes que son procesadas para obtener muchos datos relativos al usuario, entre los que destacan la posición y rotación de hasta 25 articulaciones diferentes, formando un esqueleto completo que permite definir o identificar cualquier movimiento que el jugador realice. La transmisión de estos datos al ordenador se realiza a través de un puerto serie USB (Universal Serial Bus). Así pues, la función del middleware es recibirlos, empaquetarlos y enviarlos a Unity mediante un socket de transmisión interno. Dicha comunicación se realiza en tiempo real, por lo que se utilizó UDP (User Data Protocol), un protocolo de transmisión que prima la velocidad de la transmisión ante la seguridad de la recepción.

En la figura 2 [12] se representa el sistema de transmisión diseñado.

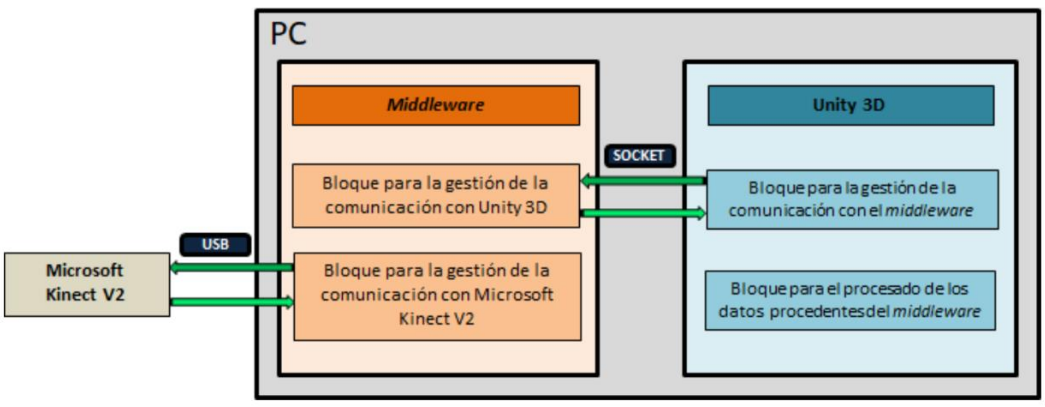

Figura 2. Diagrama de bloques del sistema Kinect – K2UM – Unity 3D.

Los datos obtenidos del middleware se gestionan internamente en Unity con un *GameObject* llamado *KinectAsset*. Dicho elemento se compone de un esqueleto con las articulaciones que es capaz de detectar la cámara y con sus relaciones de jerarquía, es decir, si se levanta la rodilla, el resto de la pierna se mueve consecuentemente. Este esqueleto se podrá asociar a cualquier elemento del juego y así poder controlarlo a partir de los movimientos del usuario. Para que esto sea posible, el middleware incorpora una serie de scripts que analizan los datos obtenidos del middleware y aplican las rotaciones y traslaciones pertinentes al esqueleto mencionado.

También incorpora un módulo de amplificación del movimiento, mediante el cual se puede configurar que la salida, es decir, el movimiento aplicado al esqueleto, sea amplificada respecto a la entrada, los datos captados por la Kinect o, dicho de otra manera, el movimiento del usuario.

2.3. Buscando a Totoro

El punto de partida de este proyecto es el videojuego *Buscando a Totoro (BaT)* [13], desarrollado para la asignatura Síntesis y Animación de Imágenes por Cristina Corporales, Angélica Suárez y Pablo López, el autor de esta memoria.

2.3.1. Historia

Buscando a Totoro es un juego de plataformas 3D en el que el protagonista Fruitman, un personaje hecho de frutas, tiene que salvar a su amigo, quien se encuentra retenido en el planeta CoKu Tau 257, donde habitan los malvados Yokais. La misión de Fruitman es buscar a su amigo y escapar juntos.

La acción transcurre en un laberinto circular que pretende simular el interior del tronco de Yggdrasil, árbol de la vida y hogar de los enemigos. El objetivo de Fruitman es llegar al centro del laberinto, donde está su amigo. Una vez se encuentran, Totoro comienza a seguirlo y juntos deben salir del laberinto. El juego finaliza cuando ambos han logrado llegar a la salida sanos y salvos.

2.3.2. Mundo del juego

Al ser un laberinto obliga muchas veces a desandar el camino ya que hay ramas sin salida. De esta manera la dificultad aumenta, tanto a la hora de jugar como en el diseño. En la figura 3 se muestra una visión cenital del laberinto y se puede observar que algunos pasillos no llevan a ningún sitio. Los puntos marcados en rojo representan las posiciones desde las que nacen los enemigos.

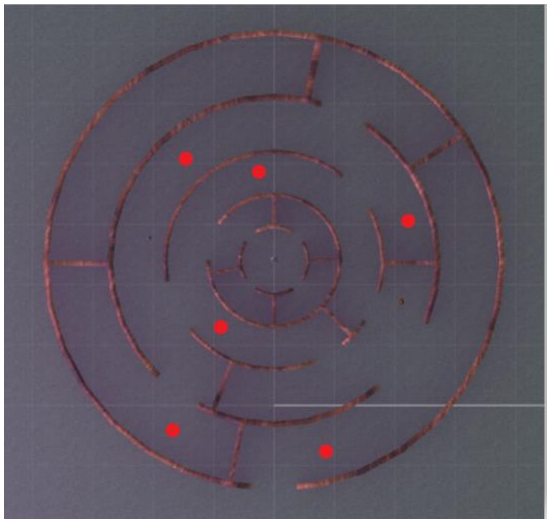

Figura 3. Diseño del laberinto.

2.3.3. Personajes

Cada cierto tiempo se genera un Yokai aleatoriamente entre una de esas localizaciones. Hay dos tipos de Yokais que se diferencian por su aspecto y la animación ya que uno de ellos tiene piernas y camina mientras que el otro se desplaza botando. Sin embargo, no tienen repercusión en la mecánica del juego ya que ambos se desplazan a la misma velocidad, atacan con la misma frecuencia y quitan la misma vida en sus ataques. La generación de uno u otro modelo también es aleatoria. Sus aspectos se representan en figura 4.

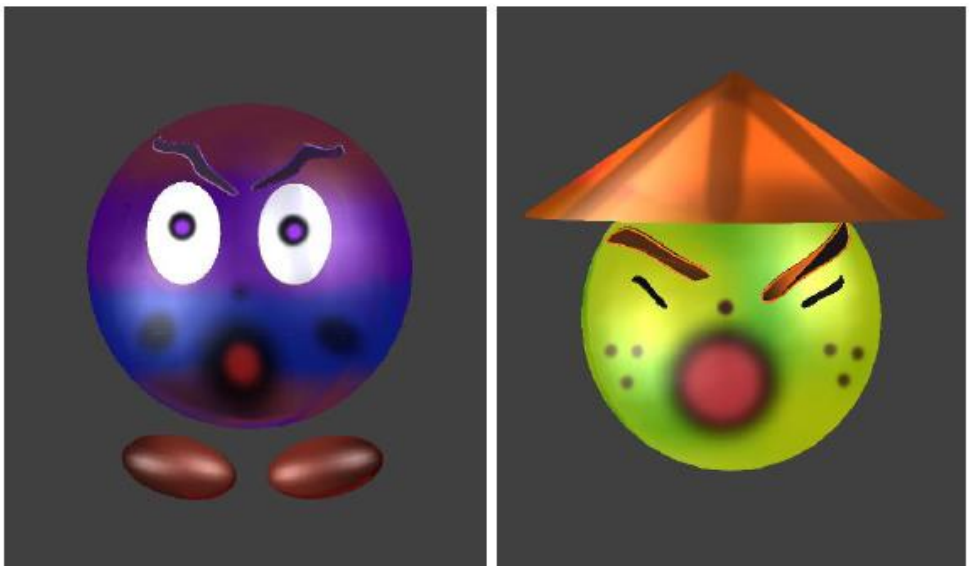

Figura 4. Diseño de los Yokais.

Se pretendía que el protagonista fuese un personaje amigable y sencillo de implementar, por lo que su diseño está basado en un clásico de los videojuegos que cumple estas características: Rayman (Figura 5), protagonista del videojuego homónimo creado por Michel Ancel y desarrollado por UbiSoft [14]. Igual que Rayman, el personaje de

Fruitman (Figura 6) está formado por una pieza principal (torso) y cinco articulaciones simples (dos manos, dos pies y la cabeza) que no tienen conexión con el torso. La diferencia con Rayman es que las piezas son frutas o verduras.

Figura 5. Diseño de Rayman [15].

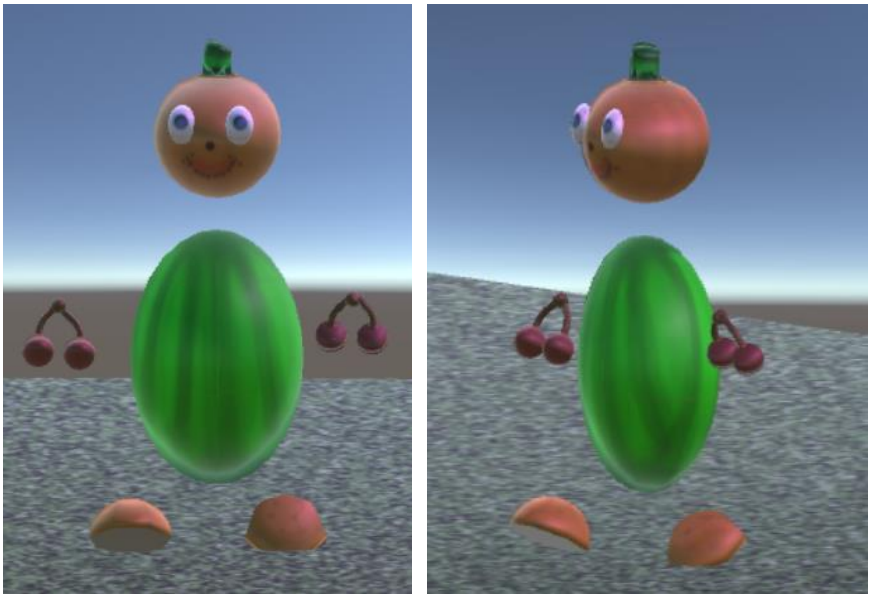

Figura 6. Diseño de Fruitman.

El último personaje clave del juego es Totoro, el amigo de Fruitman, quien está secuestrado por los Yokais. Este personaje es obra de Hayao Miyazaki, muy conocido en el mundo del cine y de la animación por la película *Mi Vecino Totoro*, de los famosos estudios Ghibli [16]. La elección de este personaje se debe al cariño que le profesaban los autores del videojuego, su modelado realizó Angélica Suárez. En la figura 7 [16] se puede observar una imagen del Totoro original y en la figura 8 una progresión del modelado de Totoro para Buscando a Totoro.

Figura 7. Diseño original de Totoro.

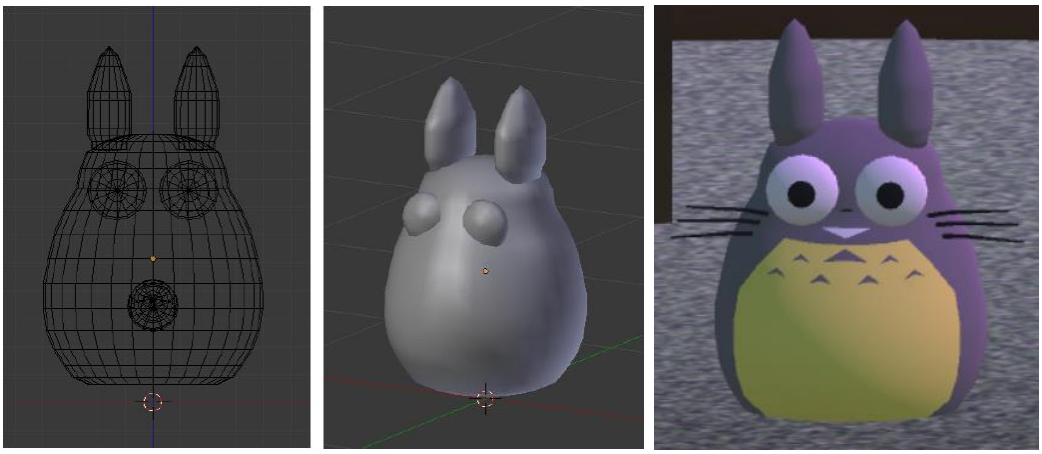

Figura 8. Diseño de Totoro.

Todos los personajes llevan una serie de animaciones que se aplican cuando están realizando determinadas acciones como caminar. Las acciones animadas dentro del juego y para cada personaje son las siguientes:

- Fruitman: inactivo (*idle*), caminar y atacar.
- Malo 1: caminar, morir.
- Malo 2: botar (caminar), morir.
- Totoro: alegrarse cuando Fruitman lo encuentra, caminar.

2.3.4. Mecánica del juego

El jugador debe controlar a Fruitman y conseguir que este llegue al centro del laberinto para rescatar a Totoro. Los Yokais están diseñados para acudir al punto donde se encuentra Fruitman y atacarle para restarle vida. Cuando los Yokais detectan que están cerca, se

dispara un ataque que le resta 10 puntos de salud a Fruitman. Su salud total es de 100 puntos, por lo que podrá soportar hasta 9 ataques sin morir. Repartidas por el mapa hay una serie de peras que Fruitman puede consumir al pasar por encima para restaurar salud. Estos consumibles regeneran 10 puntos de salud.

La manera que tiene el protagonista de defenderse será bien esquivándolos o bien atacándolos. Para mover al protagonista, el jugador deberá utilizar las teclas WASD para avanzar, girar hacia la izquierda, retroceder y girar hacia la derecha, respectivamente. Con el movimiento del ratón se maneja la cámara y con el clic izquierdo se lanzan los ataques.

Cuando Fruitman se encuentra con Totoro, se lanza una animación de alegría y aparecen unas escaleras que llevan al segundo nivel. En este nivel el protagonista aparece en el centro de un laberinto similar y Totoro comienza a seguirlo para escapar juntos. En esta fase del juego, los Yokais siguen interponiéndose en su camino para evitar que salgan, con el añadido de un temporizador que marca el tiempo límite en el cual se debe cumplir el objetivo. El videojuego termina cuando muere Fruitman, cuando el tiempo se acaba o cuando los amigos consiguen salir juntos del laberinto.

El punto de entrada al juego es una escena de menú, en la que se dispone de varias opciones: “Jugar”, “Instrucciones” y “Salir”. La pantalla de instrucciones muestra un breve texto con los controles que permiten manejar el personaje. El usuario utilizará el ratón para manejarse por estos menús, hasta que seleccione “Jugar”, momento en el cual se lanza la escena principal y comienza el juego. Dichos menús se encuentran en la figura 9:

Figura 9. Menús del juego.

3. Herramientas de desarrollo

3.1. El software

El software utilizado para este proyecto es Unity 3D [17], un motor de juegos multiplataforma para Windows, MacOS, Xbox y Playstation 4 entre otras, comercializado por Unity Technologies, San Francisco. Permite crear videojuegos tanto en 2D como en 3D para lo que se utiliza su API en C#. Su lanzamiento fue en junio de 2005 y desde entonces se ha venido desarrollando y actualizando a versiones más recientes, siendo Unity 2018.2.11 la última versión estable en el momento de redactar esta memoria y Unity 2017.2.0f3 la utilizada para el desarrollo del proyecto. La licencia básica es gratuita y suficiente para el ámbito de este trabajo. Unity ofrece una plataforma llamada *Asset Store* donde los usuarios pueden acceder gratuitamente a gran variedad de *assets*, que son objetos, texturas, músicas, scripts etc. prefabricados y de uso libre. Sin embargo, para poder comercializar los juegos creados, Unity ofrece un sistema de tarificación en base a los ingresos que genere el usuario.

El videojuego está inicialmente desarrollado con el software Blender [18]. Blender es un software de modelado y animación 3D que incorpora un modesto motor de juego llamado Blender Game Engine (a partir de ahora llamado BGE). Como se detallará más adelante, el modelo lógico de BGE se compone de sensores, controladores y actuadores conectados mediante puertas lógicas, además de soportar scripts en lenguaje Python. Sin embargo, Unity fue implementado principalmente como motor de juegos en 2 y 3 dimensiones y se basa en scripting en lenguaje C#, por lo que será necesario reescribir por completo la lógica del juego. Estas dos herramientas son perfectas para usar de forma complementaria, p.ej. creando y animando en Blender los modelos y personajes que se utilizan en Unity como elementos del juego.

Para exportar todos los elementos de Blender e importarlos a Unity es necesario un formato de intercambio común entre ambos. Esto se realiza mediante un archivo con formato .fbx [19] que permite la interoperabilidad entre entornos de captación/creación de movimiento y desarrollo en 3D. El formato FBX es propiedad de Autodesk y parte de Autodesk Gameware [20]. FBX agrupa en un archivo binario o ASCII las diferentes características de cámaras, luces, mallas o cualquier objeto que pueda usarse en una escena 3D. Así se consigue que diferentes softwares puedan usar estos datos y adaptarlos a sus entornos de trabajo.

Para la comprensión de este proyecto es necesario conocer mínimamente la interfaz de usuario de Unity 3D (ver Figura 10), por lo que se explican los módulos más importantes a continuación.

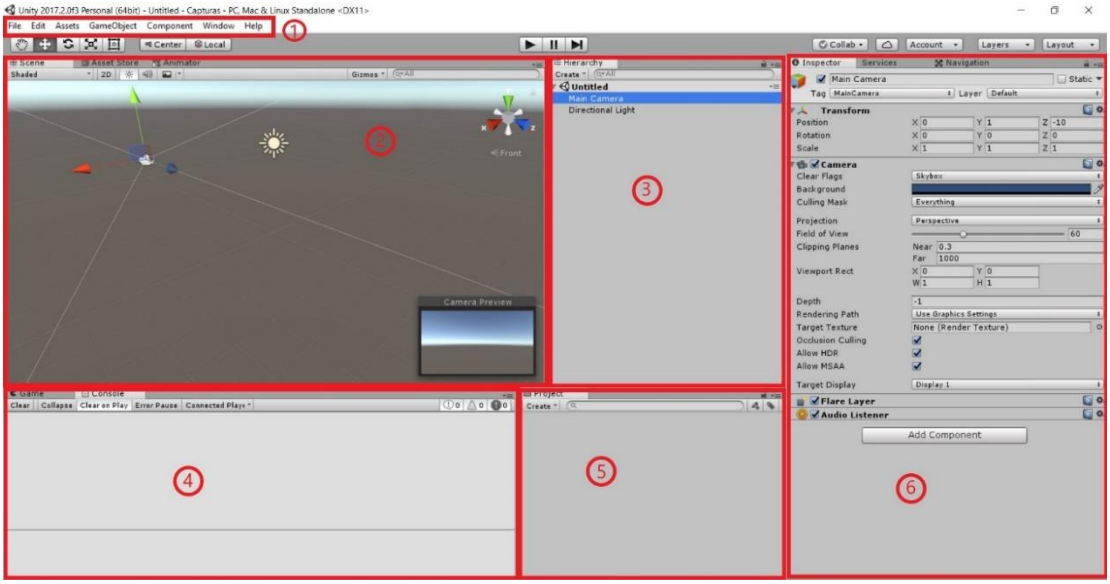

Figura 10. Interfaz gráfica de Unity.

1. **Barra de menús.** Desde esta barra se pueden realizar prácticamente todas las acciones del programa.
2. **Vista de escena.** Permite navegar por la escena y visualizar los cambios al editarla. Puede configurarse en 2D o 3D.
3. **Jerarquía.** Muestra todos los *GameObjects* que forman parte de la escena y su jerarquía. Los *GameObjects* son los elementos del juego y contienen sus componentes funcionales [21].
4. **Vista de juego/Consola.** En la vista de juego se muestra cómo se vería exactamente el juego al ser jugado, como si estuviese creado ya el ejecutable final. Sirve para probar su funcionamiento. La ventana de consola da información acerca de los posibles errores o advertencias que puedan surgir. También puede ser usada por el juego para mandar mensajes de depuración (para detectar y corregir errores).
5. **Ventana de proyecto.** Muestra todos los a*ssets* disponibles para usar en el proyecto. Cuando se importa un *asset*, aparece en esta ventana, pero el hecho de que un determinado elemento esté en esta ventana no implica que se esté utilizando en la escena.
6. **Inspector.** Permite ver y editar todas las propiedades de un *GameObject*, así como añadirle componentes y activarlo o desactivarlo.

3.2. El hardware

Para la interfaz natural de usuario es necesario capturar los movimientos realizados por el jugador para controlar el juego. Esto se realiza mediante la cámara Kinect v2 de la Xbox One de Microsoft, un dispositivo de detección del entorno compuesto por un array de micrófonos, una cámara de gran ángulo visual y una cámara de infrarrojos que combinada con un emisor de infrarrojos permite obtener un mapa de profundidad. Mediante el

procesado de los datos que estos sensores detectan se puede obtener información valiosa como la posición de hasta 25 articulaciones del cuerpo humano (como se muestra en la figura 11), expresiones faciales, latidos por minuto e incluso fuerza muscular aplicada en cada movimiento.

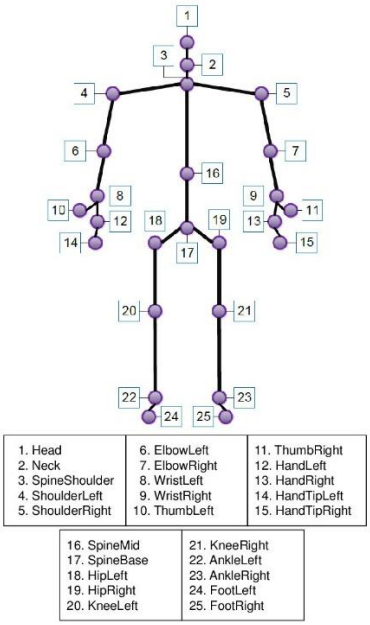

Figura 11. Articulaciones captadas por la Kinect v2 [22].

Las características técnicas son las siguientes:

- Resolución de la cámara RGB: 1920x1080 @ 30 fps.
- Resolución de la cámara de infrarrojos: 512x424 @ 30 fps.
- El campo de visión: 70 grados en horizontal y 60 en vertical.
- La distancia de detección oscila entre 0,5 y 4,5 metros.
- Con el puerto USB 3.0 la latencia es de 60 ms.

Para obtener un mapa de la profundidad de la imagen, se utiliza un sistema LIDAR (*Light Detection and Ranging*) en el que se emplea el método de TOF (*Time of flight*), por el cual se emite un haz de luz infrarroja y calcula la distancia a los objetos según el tiempo que la luz ha tardado en regresar. También es capaz de interpretar la posición de una fuente de sonido debido a su array de micrófonos. El aspecto del hardware se puede ver en la figura 12.

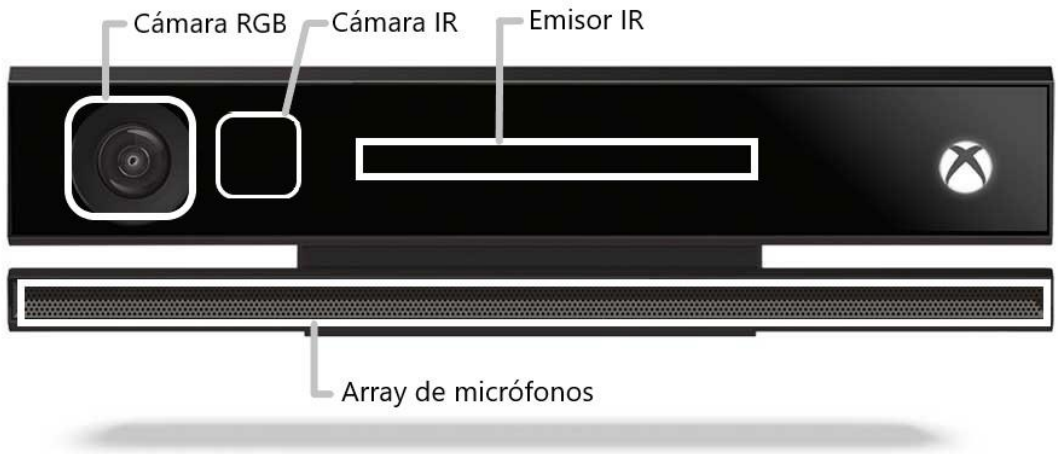

Figura 12. Kinect v2.

4. Desarrollo

4.1. Primera etapa: Traspaso de los modelos de Blender a Unity

4.1.1. Pasos previos

Previo a la fase de importación de los modelos desde Blender a Unity es necesario hacer una valoración de los elementos del juego que se quieren importar. Para ello es necesario tener cierto contexto sobre el juego del que se parte y el entorno de desarrollo Blender.

Por lo tanto, es obvio que, para seguir con la misma línea argumental, se necesitan los personajes principales Fruitman y Totoro. Además, se aprovechan los diseños de los Yokais enemigos y el escenario en el que se encuentran, que tiene forma de laberinto y en cuyo centro se halla Totoro secuestrado.

Blender es un software de modelado 3D cuya interfaz gráfica es modular y configurable. Se puede observar en la figura 13 los diferentes módulos señalados con números rojos. El 1 corresponde a la vista 3D y en ella se observan todos los cambios que se van realizando sobre los elementos del juego. El 2 es la vista del editor lógico, que permite establecer sentencias lógicas que determinan el desarrollo del juego. Los números 3 y 4 corresponden al *outliner* y a las propiedades, respectivamente. En ellos se pueden seleccionar los elementos del juego y observar sus diferentes parámetros. Todos estos módulos son configurables tanto en tamaño como en número y funcionalidad, es decir, puede haber más módulos en la pantalla e incluso varias vistas 3D o varios *outliner*.

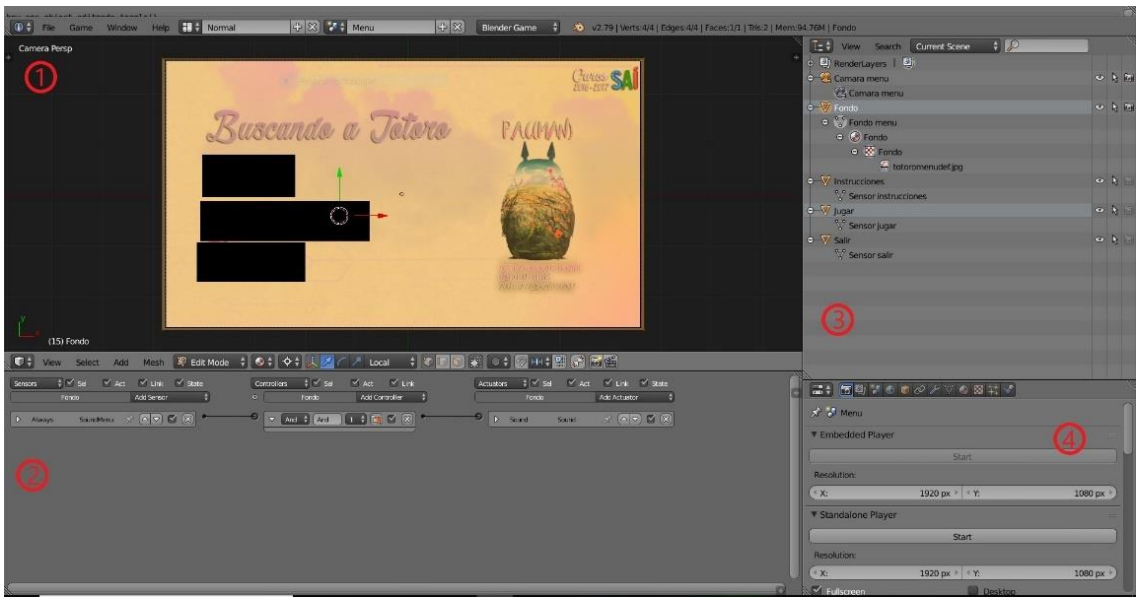

Figura 13. Interfaz gráfica de Blender.

El *outliner* de Blender presenta todos los bloques de datos y objetos que forman parte del juego. Algunos de estos objetos pueden tener una estructura jerárquica en árbol de

padres e hijos. Esto quiere decir que habrá un elemento del juego principal y otros elementos subordinados a él. Por ejemplo, en la figura 14 se puede observar que el personaje Fruitman está contenido en un elemento llamado "AvatarFruitman" y de él dependen el resto de elementos como su esqueleto, su animación o su malla y materiales. No es necesario exportar absolutamente todos los hijos de un objeto ya que el *outliner* muestra también elementos pertenecientes a Blender, como las "NLA Tracks", unas capas del sistema de animación de Blender (Non-Linear Animation) [23]. De esta manera se exportan únicamente los elementos físicos del personaje (malla, esqueleto, materiales, etc.).

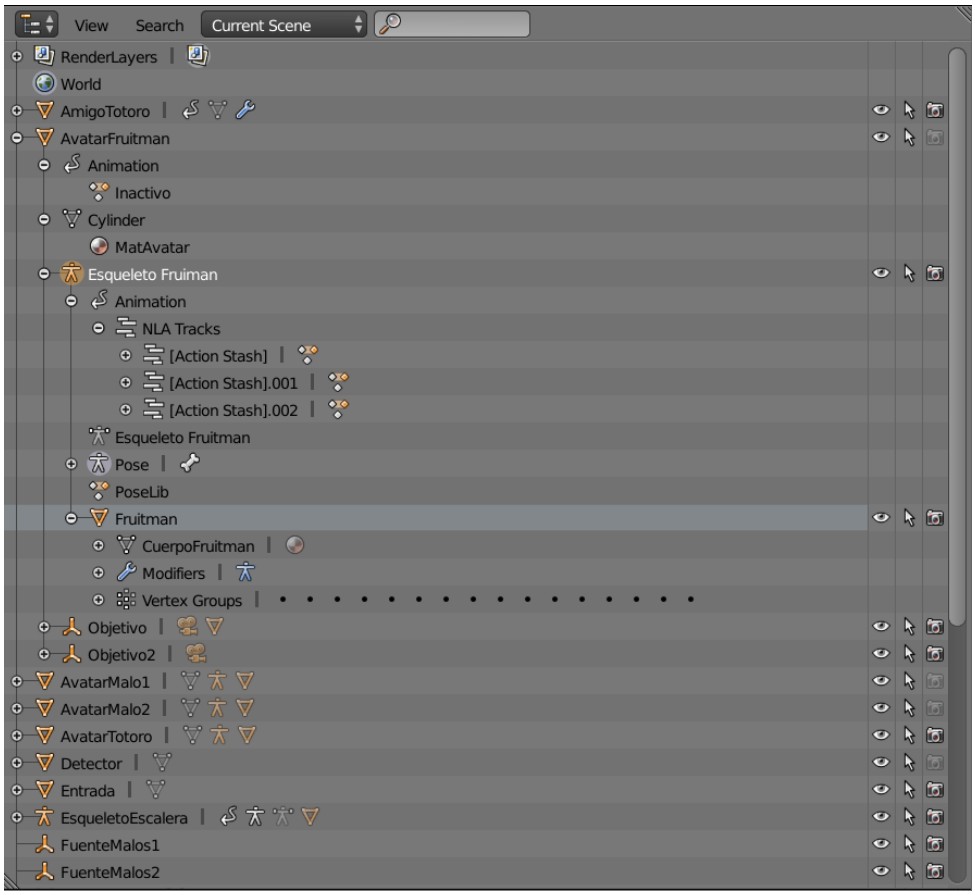

Figura 14. Ejemplo del "outliner" de Blender.

A continuación, se relatan una serie de consejos mayormente preventivos, es decir, con la finalidad de minimizar los posibles errores que puedan surgir en el proceso de exportación. No es aconsejable ignorarlos ya que puede dar muchos problemas. Más adelante se relatan algunos de los problemas más comunes.

A la hora de exportar un elemento del juego es conveniente trabajar en modo "local", un modo de trabajo que ofrece Blender y que consiste en aislar los elementos seleccionados del resto de entorno modelado, para que los cambios realizados sólo se reflejen en dichos elementos. A continuación, se detallará el proceso de exportación en el ejemplo de un elemento concreto, Fruitman. Por necesidades del proyecto en Blender, el personaje de Fruitman está contenido en un avatar, que no es más que una malla cilíndrica que rodea la malla del personaje. Este avatar sirve para simplificar los cálculos de la física, ya que es el

elemento que detecta las colisiones. No será necesario exportarlo porque Unity realiza esos cálculos de manera diferente, con un elemento propio llamado *Collider*. Lo mismo ocurre con la gestión de la cámara. Al tener ambos softwares una lógica diferente, la manera de implementar las cámaras del juego varía, por lo que ya no es necesario exportar los objetos "Objetivo" y "Objetivo2" que implementan la cámara en Blender, sino que se realizan desde cero en Unity.

Para trabajar en un entorno más limpio y evitar errores, es aconsejable crear una nueva escena dentro del proyecto como se visualiza en la figura 15: (símbolo "+" de la selección de escenas en la barra superior).

Figura 15. Creación de una nueva escena en Blender.

Una vez creada, se vuelve a la escena que contiene el personaje a exportar y se seleccionan todos los objetos que se quieran enviar a la nueva escena. Normalmente es necesario exportar todos los elementos que sean relevantes para la física del personaje. En este caso, dichos elementos son las mallas "AvatarFruitman" (no tiene relevancia física, pero es necesaria por ser el elemento padre), "EsqueletoFruitman" (por contener el esqueleto al que se aplican los movimientos) y "Fruitman" (por ser la malla del personaje). Al seleccionarlos se puede observar en la vista 3D que se remarcan sus siluetas en naranja. En este estado, se pueden copiar estos elementos a la nueva escena auxiliar (Ctrl+L (Make links)). Se abrirá un menú desplegable y se seleccionará la nueva escena en el submenú "*Objects to scene*".

Para evitar problemas de referencias espaciales, es necesario llevar el objeto al origen de coordenadas global (OG). Si esto no se cumple, hay un *offset* entre el origen de coordenadas del objeto (OO) y OG. Esta diferencia ocasiona problemas en el programa destino ya que en la exportación OG pasa a ser OO, es decir, al importar el elemento en un punto concreto de Unity, su malla y su esqueleto aparecen en otro punto, alejado una distancia igual a ese *offset* inicial.

Por tanto, se debe asegurar que la transformación del objeto a exportar, es decir, su origen, esté en el origen de coordenadas. Esto se consigue llevando el cursor al origen (MAYUS + S >> "*Cursor to Center"*), seleccionando el objeto y moviéndolo al cursor en el origen (MAYUS + S >> "*Selection to Cursor"*). Para comprobar que esta operación ha sido correcta se puede observar los valores de la transformación (*Transform*) en la ventana de propiedades (Figura 16) pulsando N sobre la vista 3D.

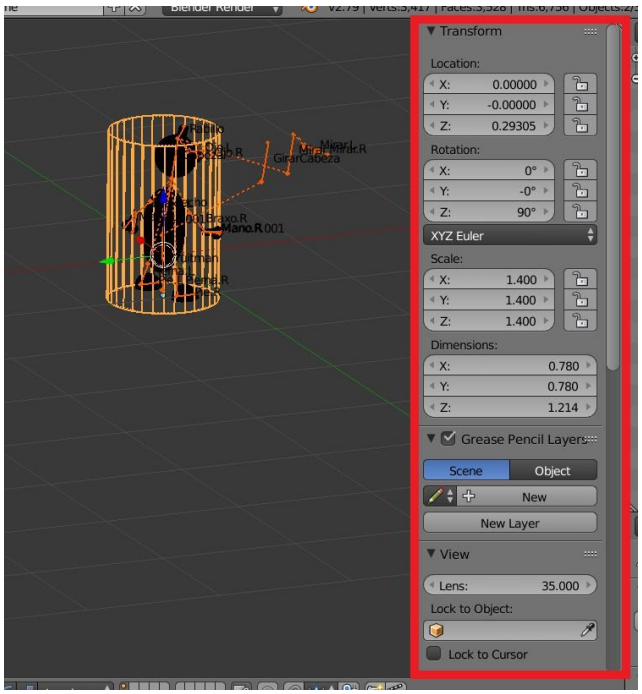

Figura 16. Ventana de propiedades en Blender.

Es necesario desvincular el avatar ya que es una malla cilíndrica utilizada en Blender para calcular las colisiones con otros objetos y, como ya se ha mencionado, en Unity estas operaciones se realizan a través de *colliders*. Para ello, se selecciona sobre el *outliner* el objeto **hijo** que se quiera separar (EsqueletoFruitman) y después (manteniendo MAYUS pulsado), el objeto **padre** (AvatarFruitman).

Seleccionados estos objetos, se borra la jerarquía entre ambos (Alt+P (*clear parent*)). Para mantener el objeto hijo en el centro de coordenadas se selecciona la opción *"Clear and Keep Transformation"*. En este punto, la jerarquía en el *outliner* debería estar como en la figura 17, con los elementos totalmente desvinculados, y ya se podría eliminar el AvatarFruitman sin que afecte a la exportación.

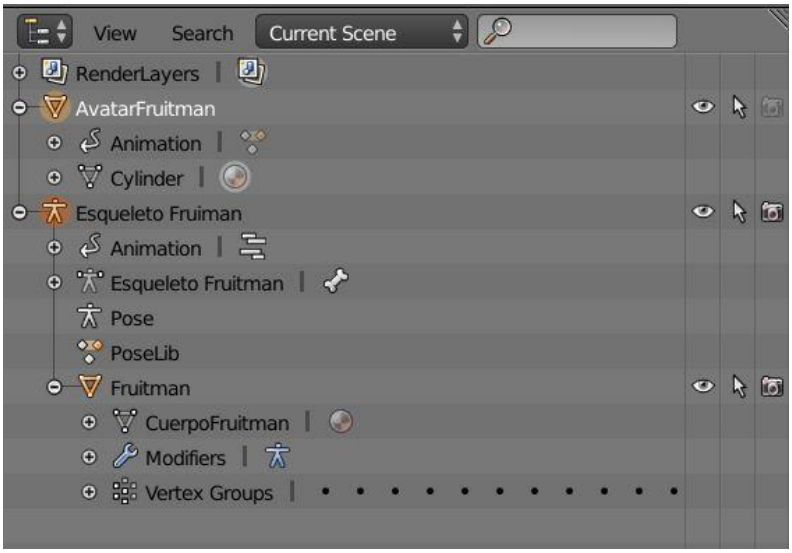

Figura 17. Jerarquía entre EsqueletoFruitman y AvatarFruitman restablecida.

Resumiendo, antes de proceder a la exportación de un objeto se debe "limpiar" de todos los elementos innecesarios y exclusivos de Blender que puedan entrar en conflicto en la exportación. Además, es muy importante tener las referencias espaciales del objeto frente a su propio origen y que este esté situado en un sitio congruente. Por ejemplo, si se trata de una persona, su origen de coordenadas lógico estaría en los pies o en la cadera, pero no en las manos ni fuera del cuerpo.

4.1.2. Exportación desde Blender

A continuación, se aborda cómo hacer la exportación desde Blender. Este software tiene diferentes opciones de ficheros, como Collada (.dae), Alembic (.abc), 3DStudio (.3ds), FBX (.fbx) o STL (.stl) entre otros. Dependiendo del destino del objeto a exportar conviene usar uno u otro. Para este caso se empleará el formato FBX. Los archivos .fbx permiten la interoperabilidad entre entornos de captación/creación de movimiento y desarrollo en 3D, lo que garantiza que el proceso de exportación de Blender e importación en Unity sea exitoso.

Por lo tanto, se debe seleccionar los elementos a exportar (EsqueletoFruitman y Fruitman) y se seleccionará *File > Export > FBX (.fbx).* La siguiente pantalla permite elegir el directorio donde guardar la exportación. Hay que prestar atención a las diferentes opciones de exportación que se encuentran en la esquina inferior izquierda (ver Figura 18).

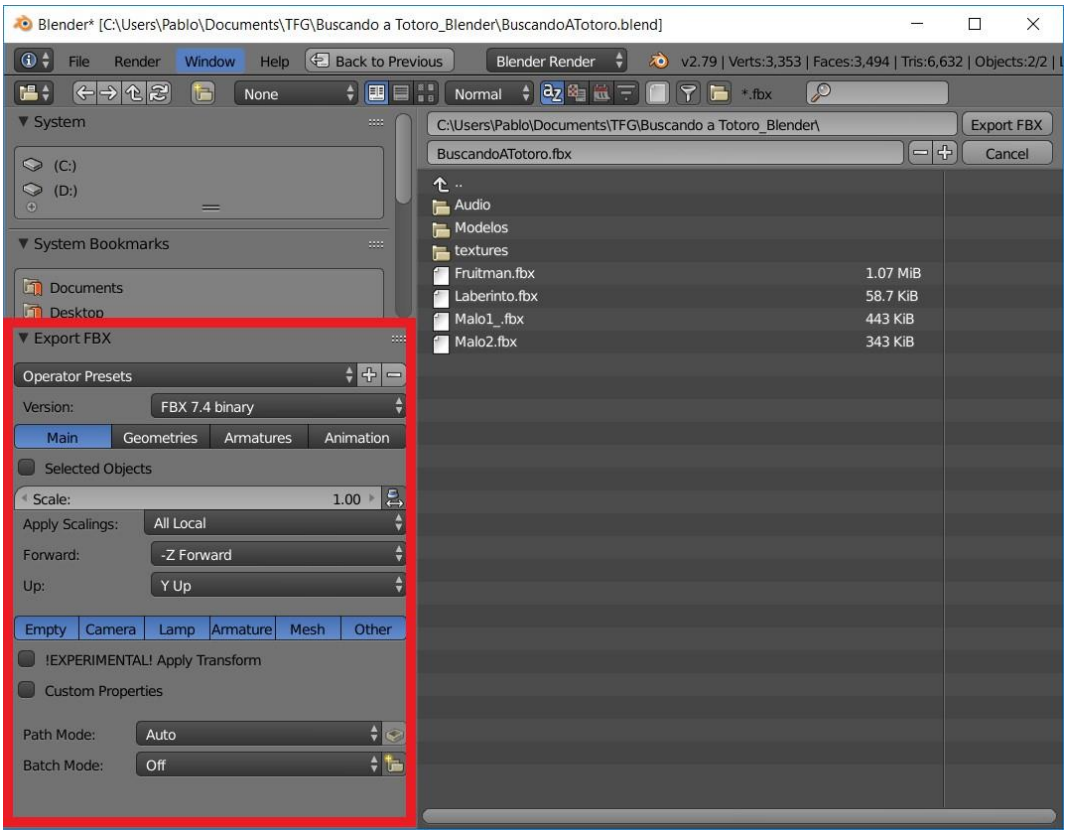

Figura 18. Opciones de exportación en formato FBX.

Este menú tiene 4 pestañas cuyas opciones se explican a continuación.

- *Main*:
    - Es importante tener **activada** la opción *Selected Objects* cuando se quiere exportar más de un elemento, como en este caso.
    - **Deshabilitar** la opción *Scale* permite no tener en cuenta las unidades de medida de Unity.
    - El desplegable *Forward* determina la dirección en la que estará orientado el personaje en el software destino (Unity). De la misma manera *Up* determina el eje vertical. Para este ejemplo las opciones por defecto son correctas y no es necesario modificarlas.
- *Geometries*:
    - Se dejan las opciones por defecto.
- *Armatures*:
    - Se **activa** la opción *Only Deform Bones*, lo cual permite que únicamente se exporten los huesos cuyo movimiento afecta o deforma la malla del personaje. Así, si existen huesos de control, serán ignorados.
    - La opción *Add leaf bones* añade un hueso adicional al final de cada cadena de huesos, por lo que se queda **desactivada**.
    - El resto de desplegables se dejan por defecto.
- *Animation*:
    - Para el personaje Fruitman en el proyecto Unity no son necesarias las animaciones, ya que el movimiento de su cuerpo viene determinado por la captación del movimiento del usuario por la Kinect.
    - Para el resto de los personajes que lleven animaciones (Yokais y Totoro), se mantienen las opciones por defecto excepto *NLA strips*, que se **desactivan** por no ser necesarias.

Finalmente se escoge el nombre con el que se quiera exportar y se selecciona ***Export FBX***. El archivo resultante ya puede importarse a Unity.

4.1.3. Importación en Unity

Se puede importar un elemento al proyecto haciendo clic derecho en la ventana de proyecto y seleccionando la opción *Import new Asset*. A continuación, se abre una ventana del explorador de ficheros en la que se debe seleccionar el fichero pertinente. Una vez hecho esto, el personaje de Fruitman aparece en la ventana de proyecto, además de una nueva carpeta *Materials* creada automáticamente y que contiene el material de Fruitman. Sin embargo, este material no se exporta correctamente en el formato .fbx y hay que realizar pasos adicionales para aplicar la textura.

En Blender se puede acceder a las imágenes que conforman la textura de un objeto en la vista de *UV/Image Editor*. Al seleccionar esta vista (1, figura 19), aparece un desplegable

(2, figura 19) con las imágenes disponibles en el proyecto y al seleccionarlas, se muestran en el espacio reservado para ello (3, figura 19).

Figura 19. UV/Image Editor Blender.

Para extraer la textura de la imagen seleccionada basta con abrir el menú *Image* situado a la izquierda del selector de imágenes y hacer clic sobre la opción de *Save as Image* (o F3). Lamentablemente hay que repetir este proceso para todas las imágenes de las que se quiera obtener textura, ya que la opción *Save all Images* arroja frecuentemente un error (*Invalid path*).

Desde Unity es conveniente crear un directorio *Textures* en la ventana de proyecto para mantener una buena organización. Este directorio contiene todas las imágenes y se pueden arrastrar directamente. También se pueden importar con Clic dcho. > *Import New Asset.*

A continuación, se arrastra el *GameObject* Fruitman desde la ventana de proyecto a la jerarquía o a la ventana de escena. Como ya se preveía, el personaje se ve gris y sin textura. Para aplicar una textura a un *GameObject* en Unity es necesario que la malla de este disponga de un material, como se observa en la figura 20.

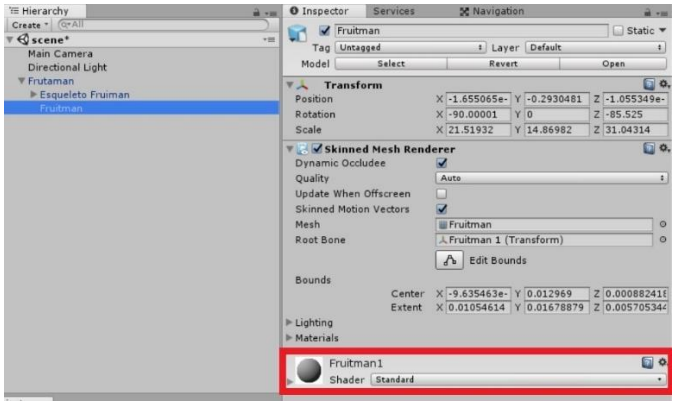

Figura 20. Material del GameObject Fruitman.

Finalmente, se selecciona la textura deseada en la ventana de proyecto y se arrastra encima del *GameObject* que contenga el material del personaje. No es estrictamente necesario que el objeto raíz o padre contenga todas las propiedades del personaje. En este caso, al hacer la importación desde el .fbx, la malla y el esqueleto se han dividido y pasan a ser dos objetos dependientes del padre, un *GameObject* con únicamente un componente *Animator*.

4.2. Segunda etapa: Generación de la lógica

4.2.1. Blender vs Unity

El modelo lógico empleado por BGE se basa principalmente en una interfaz gráfica apoyada por scripts en Python. Este documento se centra en la interfaz gráfica, ya que *Buscando a Totoro* únicamente implementaba un script de control de la cámara en Python. Dicha interfaz gráfica consta de tres bloques lógicos: sensores, controladores y actuadores, como se puede ver el la figura 21.

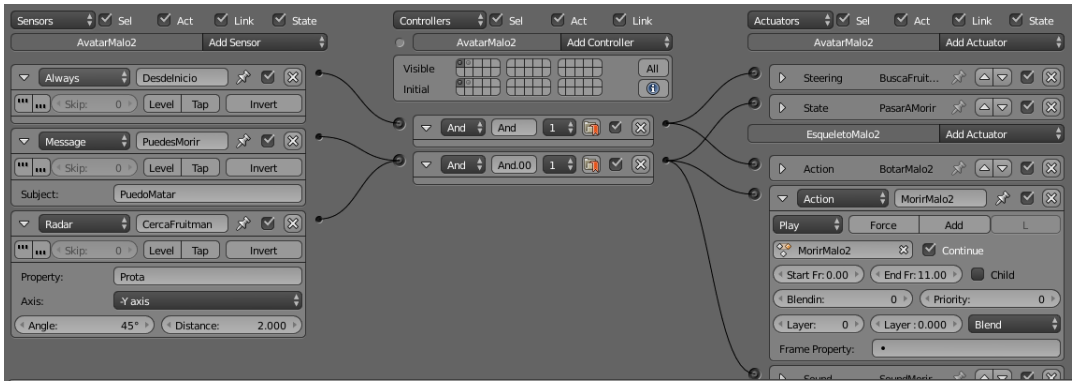

Figura 21. Modelo lógico de BGE.

Simplificando, los **sensores** detectan algún cambio en el juego (como una colisión, una tecla pulsada, etc.) y generan una respuesta lógica, es decir, un pulso positivo a dichos eventos. Los **actuadores** realizan acciones sobre los elementos del juego, como mover al personaje, lanzar animaciones o reproducir clips de audio. Finalmente, los **controladores** son el nexo entre los componentes ya mencionados. Reciben las diferentes respuestas de los sensores y se encargan de proporcionar una respuesta que será enviada a los actuadores. En este contexto los controladores son funciones o expresiones lógicas que determinan cuándo se disparan los actuadores.

Sin embargo, la filosofía de trabajo de Unity es radicalmente distinta. La unidad básica dentro de un juego se denomina *GameObject*. Cada uno de estos objetos puede tener uno o varios componentes asociados, desde una malla a una fuente de audio o un script. Además, estos elementos pueden estar agrupados jerárquicamente y tener dependencias entre sí.

En la parte que atañe a la lógica del juego, los componentes más importantes son los scripts en C#. Estos determinan el comportamiento de cada *GameObject* ante diferentes estímulos o eventos del juego, lo que implica, como se verá más adelante, una reprogramación de Buscando a Totoro casi desde cero.

La manera que tiene Unity de gestionar las entradas, los eventos e interacciones y hacer que los elementos del juego interactúen entre ellos es mediante scripts. El software utiliza MonoDevelop, un entorno de desarrollo integrado, aunque también tiene características propias para poder acceder al motor de juego desde los scripts. Como lenguajes de programación, se utilizan C# y UnityScript (basado en JavaScript). Sin embargo, en este proyecto se utiliza VisualStudio como entorno de desarrollo y los scripts son únicamente C#.

Los scripts se utilizan como componentes que permiten programar o modificar el comportamiento de los diferentes *GameObjects*. Un script se puede crear de diferentes maneras, pero si se tiene claro a qué *GameObject* va a ir asociado, la más rápida es desde el inspector, *"Add component > (Type) Script"* y darle el nombre que se desee. Es necesario que el nombre empiece con una letra mayúscula para mantener la convención de nombres en programación.

La clase básica de la que derivan todos los *scripts* en Unity es MonoBehaviour. Las funciones más importantes y habituales que se usan para el desarrollo del juego son las funciones de inicialización (*Start*), actualización (*Update*, *FixedUpdate* y *LateUpdate*) y las de gestión de eventos (*OnTrigger*, *OnCollision*) [21].

- *Start*(): Se ejecuta nada más iniciar el juego, por lo que se utiliza siempre que se quiera inicializar una variable o acceder a otro componente del *GameObject* para tener una referencia local. La función *Awake*() funciona de manera parecida, ya que también se ejecuta al iniciar. La diferencia reside en que la ejecución de la función *Start*() depende de que el componente script esté activo en el momento de iniciarse el juego.
- *Update*(): Los videojuegos son objetos 3D a los que se les aplican fuerzas y animaciones, por lo que su aspecto y/o posición va variando y se recalcula cuadro a cuadro. Esta función se encarga de ello, ya que se ejecuta justo antes de que cada cuadro sea renderizado. Su uso es el más común a la hora de desarrollar videojuegos. Las siguientes funciones son variaciones de esta:
    - *FixedUpdate*(): Se emplea para los cálculos de física (colisiones, fuerzas, etc.). No tiene la misma frecuencia que Update() ya que puede ejecutarse varias veces cada frame o no ejecutarse si no hay cálculos de física involucrados.
    - LateUpdate(): Es útil para hacer cambios que requieran que ya se hayan calculado todas las animaciones porque es la última que se ejecuta. Un ejemplo serían los movimientos de la cámara.

- *OnTrigger*: Es una familia de funciones que se ejecutan siempre que se "dispare" un indicador al chocarse dos componentes *collider* si uno de ellos está habilitado como *trigger*.
    - *OnTriggerEnter*(): Se ejecuta una vez cuando los objetos chocan.
    - *OnTriggerStay*(): Se ejecuta cada frame que la condición de choque sea verdadera.
    - *OnTriggerExit*(): Se ejecuta cuando los objetos dejan de estar en contacto.
- *OnCollision*: Es otra familia de funciones de los componentes *collider*. Es análoga a la anterior, siendo la única diferencia que ninguno de los *collider* está habilitado como *trigger*.

Estas funciones se utilizan como hilo conductor del código. Por ejemplo, la función *Update*() es necesaria en la mayoría de *scripts* ya que dentro de ella se debe escribir el código que se ejecute cuadro a cuadro.

4.2.2. Menús y Gestión de escenas

La estructura de las escenas y las transiciones entre ellas es heredada de *Buscando a Totoro* y se detalla en la figura 22. Al iniciar el juego se despliega el menú principal, que da acceso al resto de escenas. Aquí, el jugador puede informarse sobre las instrucciones antes de jugar. Por facilidad de uso, la navegación entre las diferentes escenas que no forman parte del nivel principal se realiza mediante el ratón.

Una vez el jugador selecciona "Jugar" en el menú principal, es dirigido a la escena donde se encuentra el juego. En este momento sólo hay dos posibles resultados: ganar o perder. De esta manera dependiendo de lo que suceda en el transcurso del juego se lanza una u otra escena, con música triunfal y una imagen alegre si se ha ganado o música lenta y una imagen de Totoro triste si se ha perdido. Ambas escenas tienen un botón para volver al menú principal por si se quiere volver a jugar.

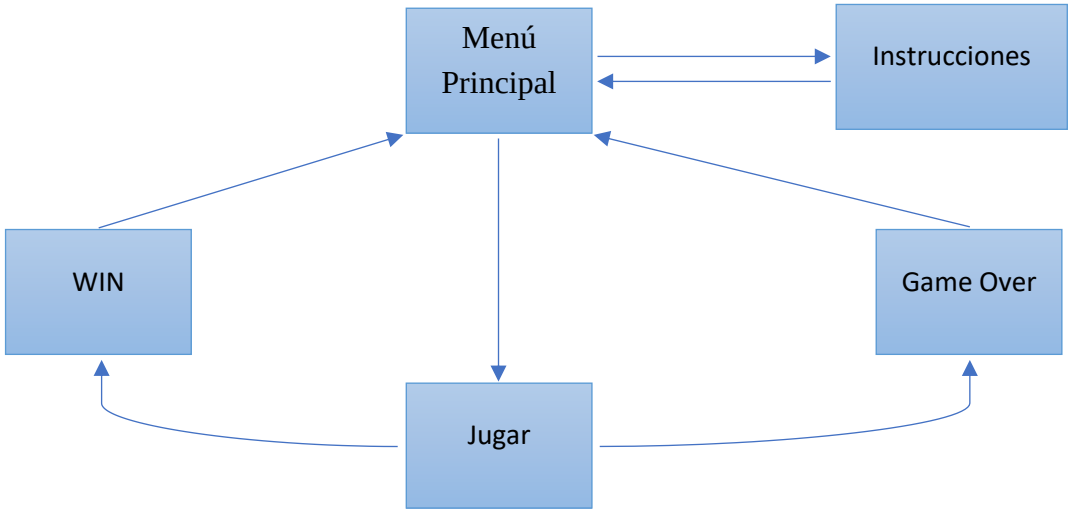

Figura 22. Escenas y transiciones del juego.

La manera más cómoda de crear escenas es hacerlo directamente desde la ventana de proyecto. Es una práctica habitual y recomendada crear una nueva carpeta llamada *Scenes* para organizar el sistema de archivos del proyecto, donde irán todas las escenas. Para crear una nueva escena se seguirán los siguientes pasos: Botón derecho sobre la carpeta *Scenes* > *Create* > *Scene* y se nombra como se desee. Una vez estén todas las carpetas creadas, se asigna un índice, como se puede observar en la figura 23, para diferenciarlas y manejarse entre ellas desde *File* > *Build Settings* (o Ctrl. + Shift + B).

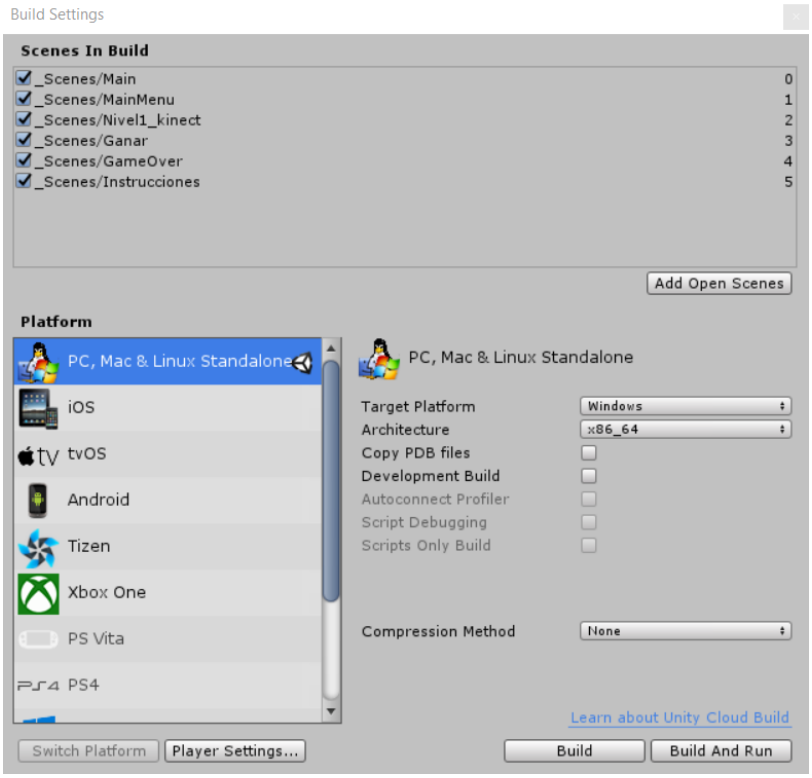

Figura 23. Ventana de Build Settings en Unity.

Al pulsar sobre el botón *Add Open Scenes*, se añade la escena actual a la lista y se asigna el índice mencionado. Este índice es utilizado posteriormente como parámetro de las funciones que permitan saltar entre escenas dentro del juego, como por ejemplo *LoadLevel*(). Se puede alterar el orden arrastrando las escenas.

Las primeras escenas que aparecen en el juego son menús, es decir, imágenes estáticas que contienen botones para interactuar con ellas, concretamente botones asociados a ciertas partes del texto como "Jugar" de tal manera que si se hace clic sobre ellos, se realizan diferentes acciones como lanzar una determinada escena. Para gestionar estos menús hay que crear elementos de UI (*User Interface*), como el *canvas* (lienzo), que es el *GameObject* que agrupa las diferentes imágenes y botones necesarios. También tiene asociados los scripts que manejan los cambios de escena. En la figura 24 se muestra la escena Main con el *GameObject* "Canvas" seleccionado, en cuyo editor aparecen sus diferentes componentes, entre los que destacan la imagen asociada de inicio de *El Desafío de Fruitman* (Image (Script)) y el *script* que gestiona los cambios de escena (Scene Man

(Script)). El Canvas tiene como hijo un *GameObject* "*Button*", que abarca la totalidad de la escena, como se puede observar en la vista de escena de la figura 24 mediante los selectores azules que contienen toda la imagen.

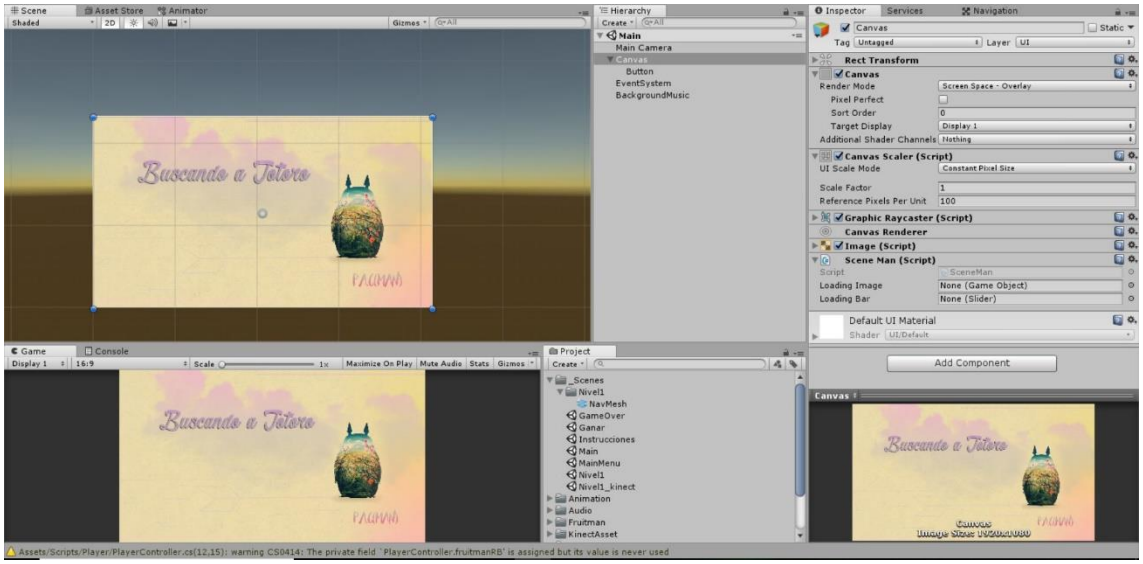

Figura 24. Ejemplo de una escena (Main) en la que sólo intervienen elementos de UI.

Los botones en Unity contienen un componente como el mostrado en la figura 25, que permite especificar la acción a realizar cuando son pulsados. En este caso, al pulsar el botón, se llama a la función *LoadLevel(int)* del *script* "SceneMan". El número uno que aparece en el campo inferior derecho es el argumento que se le pasa a la función, en este caso para cargar el menú principal. Otro ejemplo de esta implementación es el botón "Jugar", que llama a la función "CargarAsync" que, mediante una operación asíncrona, permite mostrar una barra de progreso mientras un hilo en segundo plano carga el nivel principal.

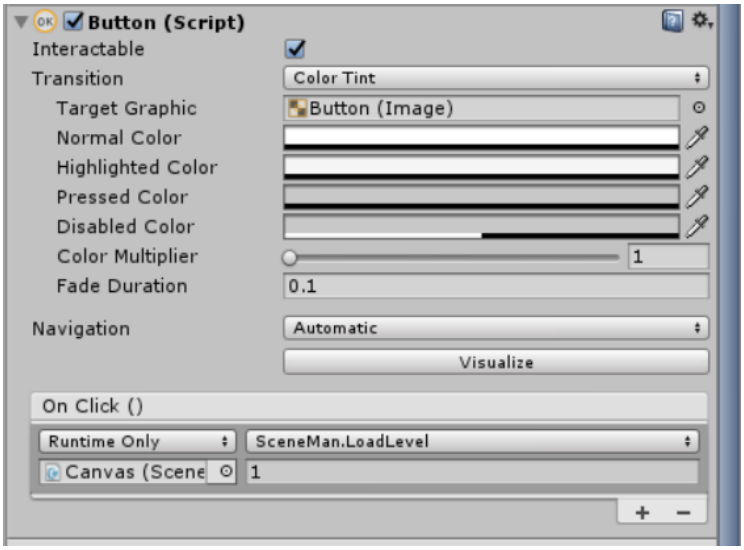

Figura 25. Detalle del componente botón asociado al GameObject Button.

Esta escena *Main* contiene además un *GameObject* llamado *Background music* (figura 26) que gestiona la reproducción de música durante el juego mediante el script

"CambiarMusica". Dicho script recibe el índice del nivel que se acaba de cargar y reproduce uno u otro clip de audio según convenga. Para ser capaz de hacer eso se necesita que ese elemento del juego permanezca siempre activo y en escena, lo cual se puede conseguir mediante la función de Unity *DontDestroyOnLoad(gameObject)* que evita que los elementos de la escena anterior sean destruidos al cargar una nueva.

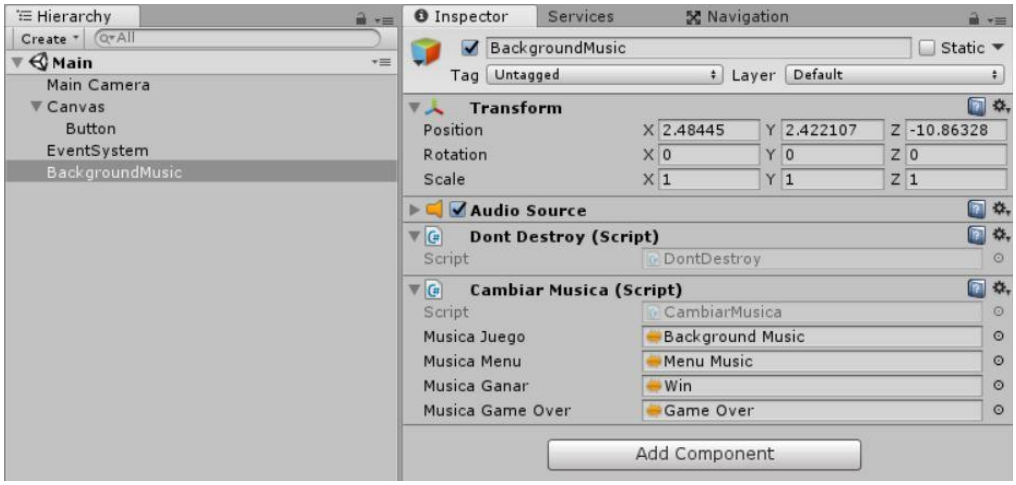

Figura 26. Componentes del GameObject "BackgroundMusic".

Un elemento importante en la escena principal (*Play*) es el gestor de la puntuación, un *display* que muestra cuántos enemigos han sido derrotados hasta el momento. Para ello se debe crear un elemento de UI tipo texto con un *script* asociado que contenga una variable numérica con el marcador. El resultado se puede ver en la esquina superior derecha de la figura 27. Esta variable debe actualizarse a cada cuadro por lo que es necesaria una función pública, accesible desde otros *scripts* para modificar su valor cuando un enemigo muere.

Figura 27. ScoreManager.

De manera análoga se crean las escenas *MainMenu*, Instrucciones, Ganar y *GameOver*, recayendo el gordo de la ejecución del juego en la escena principal, *Play*.

4.2.3. Acciones de los personajes

En el juego intervienen 3 elementos importantes: los enemigos, el protagonista Fruitman y Totoro. Estos elementos interactúan entre sí estableciendo unas relaciones entre ellos que permiten el desarrollo del juego. Por un lado, la relación entre Fruitman y los Yokais es antagónica, son enemigos. Esto implica que tienen que luchar y habrá ataques cuando estos se encuentren en el mundo del juego. Por otro lado, Totoro espera que su amigo Fruitman lo salve, es decir, tienen una relación de amistad. Por ello, cuando se encuentren debe ser un momento alegre.

Movimiento de los enemigos

La lógica que requieren los personajes es más compleja que la de las escenas y es necesario utilizar elementos como referencias, inicializaciones y funciones de gestión de eventos. A continuación, se explica el comportamiento de los enemigos. En la figura 29 se muestran los componentes necesarios, que son:

- *Transform*: necesario en todos los *GameObjects*. Indica su posición en el mundo, su rotación y su escala.
- *Animator*: contiene una referencia al *Animator Controller*. Sobre este componente se incidirá más adelante.
- *Rigidbody*: se utiliza en los cálculos de física. También se activan las opciones de restricciones (*constraint)* para que únicamente se pueda mover por el plano XZ, esto quiere decir que los Yokais únicamente pueden moverse a nivel de suelo.
- *Colliders*: se emplea para detectar la colisión con Fruitman y, a través del script, actuar en consecuencia. El *Sphere Collider* está marcado como *trigger*. Al seleccionar esta opción, el espacio delimitado por el *collider* puede ser atravesado por otros objetos, ya que únicamente detecta la incursión y genera un pulso lógico en consecuencia.
- *Audio Source*: este componente tiene asociado un clip de audio que se reproduce desde un script.

  Nav Mesh Agent: *NavMesh* es un componente de Unity que permite mapear cualquier área del juego mediante polígonos simples y generar caminos entre dos puntos de dicho mapa, como se muestra en la figura 28.

  De esta manera, se puede asignar como posición destino la posición en la que se halla Fruitman y como agente de navegación (*NavMeshAgent*) a los propios Yokais. Internamente Unity realiza cálculos para obtener las rutas y evitar los obstáculos que pueda haber en el camino, como el laberinto en este caso concreto.
- *Scripts*: determinan cómo se comportan los enemigos.

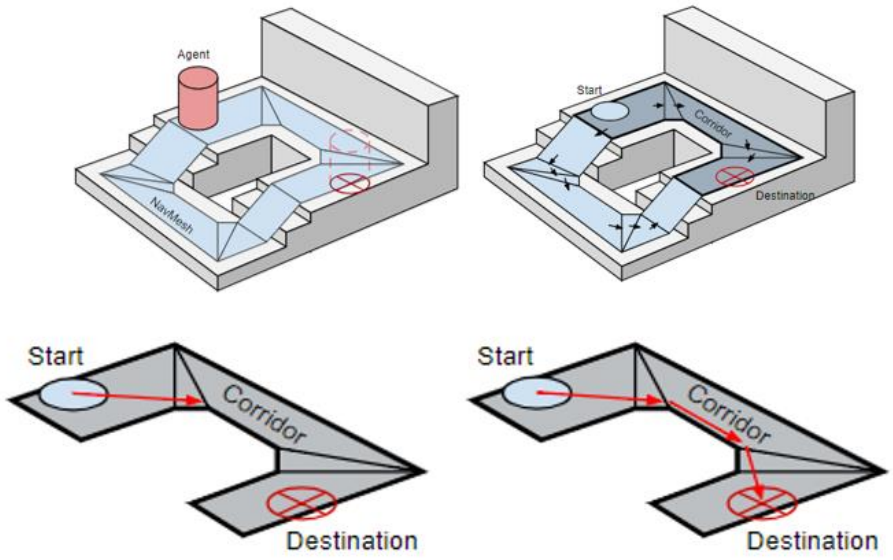

Figura 28.Funcionamiento a alto nivel del componente NavMesh [21].

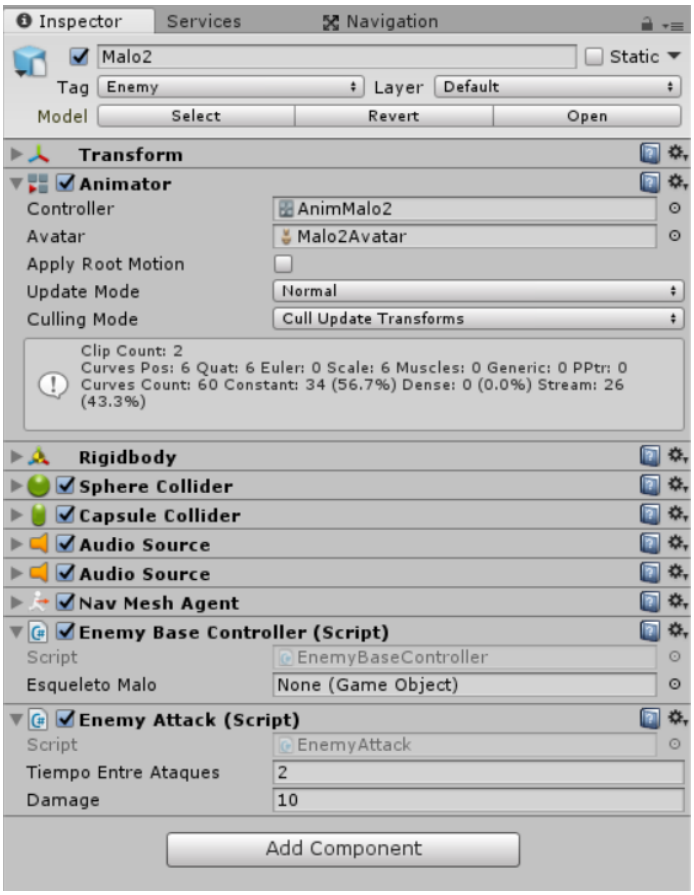

Figura 29. Componentes de Malo2 (Yokai enemigo).

A continuación, se explica en detalle cuál es la función de dichos *scripts*. Los enemigos llevan dos asociados, uno que gestiona el movimiento y otro que gestiona los ataques y la vida de los Yokais. Ambos necesitan tener referenciado a Fruitman porque, a nivel lógico,

en todo momento los enemigos necesitan conocer tanto la posición como la salud del protagonista y esto se logra "conectando" los componentes de cada *GameObject* mediante referencias.

Por lo tanto, para gestionar el movimiento de los enemigos, *EnemyBaseController* debe contener referencias a la *Transform* de Fruitman y al su propio componente *NavMeshAgent*. De esta manera, se consigue pasar la posición de Fruitman al componente de navegación y permitir que el enemigo avance siempre hacia este punto del mapa mediante la función *SetDestination(position)* de *NavMeshAgent* y controlar el desplazamiento de los Yokais.

Ataque de los enemigos

Los Yokais no tienen armas ni ningún elemento que permita atacar a distancia, por lo que deberán atacar cuando estén cerca de Fruitman. Para ello, *EnemyAttack* debe detectar las colisiones con el protagonista, comprobar su salud para ver si no está muerto ya y, si está vivo, atacar. De la misma manera, también debe recibir los ataques de Fruitman y gestionar su muerte destruyendo el *GameObject* del Yokai e incrementando el contador de puntuación. Para esto es necesario tener un *collider* que detecte las colisiones con el protagonista, una referencia al script que gestiona la salud de Fruitman y otra al *ScoreManager*.

Al detectar una colisión con Fruitman espera dos segundos (modificables) mediante una corrutina antes de atacar. Esto es necesario para dar al usuario un margen de tiempo para que logre atacar al Yokai. Si no se hiciese esto, el enemigo le restaría salud a Fruitman en el momento que se detectase, sin permitir que Fruitman, cuyo ataque es controlado por el usuario y por ello es más lento, pudiese hacer nada. Una vez pasa este tiempo, el *script* llama a la función RecibirDaño() del *script* que controla la salud de Fruitman, *PlayerHealth*. Para que no ataque continuamente se crea una variable de control "tiempoEntreAtaques" (modificable) que determina el tiempo en segundos que debe pasar entre cada ataque. Esto se hace mediante un contador.

Las animaciones se rigen por un *AnimatorController*, elemento que hay que crear en la ventana de proyecto bajo una carpeta *Animation* para mantener el orden. Este elemento es el que gestiona las animaciones y debe ir ligado al componente *Animator* del *GameObject*. Una vez asociado, si se hace doble clic sobre él, se abre una nueva pestaña en la vista de escena llamada *Animator*. Desde esta pestaña se decide el flujo de la animación, es decir, qué animación se lanza por defecto, cuál es la siguiente, qué condiciones se deben cumplir para la transición entre ambas, cuál es el tiempo de transición, etc. Por defecto contiene dos estados: *Entry*, que determina el punto de entrada nada más arrancar el juego, y *AnyState*, que permite definir una transición desde cualquier estado. Esto es útil para las muertes, por ejemplo, ya que da igual lo que el personaje esté haciendo en un momento dado porque si le matan, tendrá que lanzarse directamente la animación de morir.

Para añadir un módulo de animación a esta pestaña se arrastrará desde la ventana de proyecto. Las animaciones se guardan como hijos del elemento que se ha importado, con un símbolo de *play* al lado del nombre. Desde el asistente de importación en el inspector, se pueden realizar ciertos ajustes como modificar la duración en *frames* de las animaciones o establecer loops.

Por defecto, cuando se arrastra una animación a la ventana del animador, se crea una transición de *Entry* a la nueva animación. Esto implica que es la que se va a ejecutar inicialmente. Es obligatorio tener un estado inicial ya que si no la máquina de estados no sabe por dónde empezar. Una vez se tengan todas las animaciones que vayan a entrar en juego será necesario crear las transiciones. El estado descrito hasta ahora queda reflejado en la figura 30.

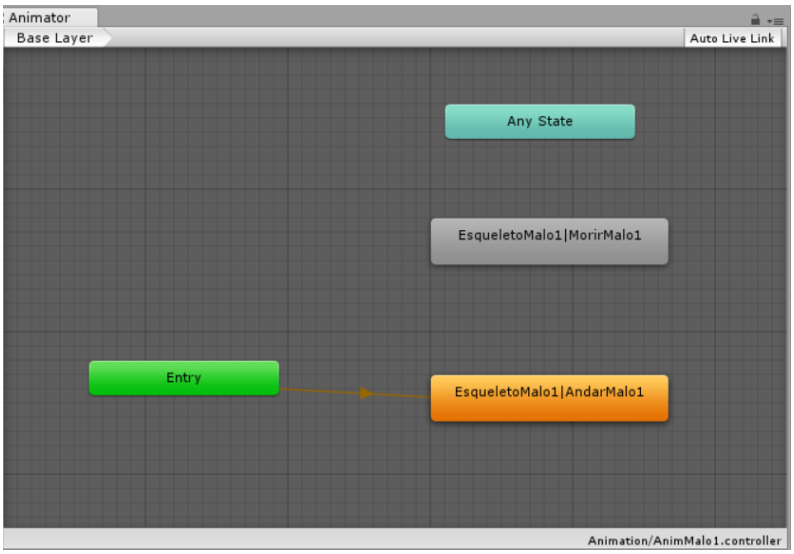

Figura 30. Ejemplo de la ventana Animator.

Las transiciones permiten definir cómo saltar de una animación a otra. Para crearlas hay que hacer clic derecho en la animación inicial, es decir, de la que se quiera partir, y seleccionar *MakeTransition*. Aparece una flecha que indica la dirección de la transición y queda definida haciendo clic sobre la animación final. Dentro de las transiciones se puede ajustar el tiempo que tarda en hacer el cambio y la cantidad de mezcla entre ambos.

Para que una transición se ejecute, es necesario que se cumpla o se deje de cumplir una condición. Se deben definir los *Parameters*, que pueden ser del tipo *Float*, *Int*, *Boolean* o *Trigger*. En el ejemplo anterior se debe definir un parámetro que indique si está caminando o no (*Bool IsWalking*) y posteriormente se asigna en el inspector de la transición, bajo el menú *Conditions*.

Ataque y vida de Fruitman

El otro elemento importante del juego en cuanto a lógica se refiere es el personaje de Fruitman. Es necesario gestionar cómo interactúa con los enemigos. Para interactuar con ellos debe estar cerca ya que tampoco tiene armas a distancia, además, se debe tener en

cuenta que el personaje tiene una vida limitada y, por tanto, es necesario gestionarla en todo momento. Por ello Fruitman tiene tres *scripts* principales: *PlayerHealth*, que gestiona la vida del protagonista, *PlayerController*, que gestiona sus movimientos y *FruitmanAttack*, que gestiona los ataques. *PlayerController* se explica en el siguiente apartado ya que para controlar los movimientos del protagonista es necesaria la intervención de la cámara Kinect.

Por tanto, *PlayerHealth* deberá tener una función pública que permita a otros *scripts* del juego, en este caso *EnemyAttack*, enviar la información de que Fruitman ha recibido un ataque. En ese momento se genera un flash rojo que simula la recepción del daño y se actualiza la barra de vida. Se comprueba también la salud actual y si esta ha llegado a cero, se gestiona la muerte de Fruitman. Su muerte consiste en destruir su *GameObject* y un cambio a la escena de *GameOver*.

FruitmanAttack debe gestionar los ataques de Fruitman. Para realizar dichos ataques, el usuario debe subir y bajar los brazos, que, gracias a la Kinect, irán en consonancia con las manos del protagonista. Por ello se decide tratar esta lógica con dos componentes *colliders* llamados *AttackTargets* como se ve en la figura 31.

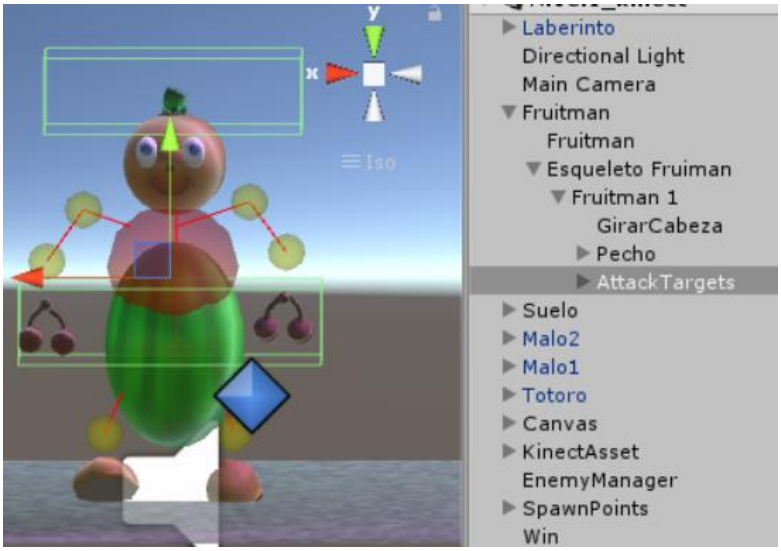

Figura 31. Colliders AttackTargets.

Estos componentes son unos paralelepípedos situados a la altura de las manos en posición de reposo y por encima de la cabeza. Detectan las colisiones con las manos de Fruitman. La idea es que el usuario solo puede realizar un ataque si lo "carga" previamente, es decir, si sube sus brazos por encima de los hombros de manera que las manos del protagonista lleguen a tocar este *collider*. Cuando esta condición se cumple, existe una variable booleana llamada <ataqueCargado> que se hace cierta. Cuando el usuario baja las manos para simular un ataque hace que otra variable <atacar> sea también cierta. De esta manera se obliga al jugador a realizar un movimiento algo exigente de brazos, ya que durante el juego es necesario eliminar a varios enemigos.

También se deben dar otras condiciones para que se realice el ataque, el Yokai debe estar cerca de Fruitman. Esto se consigue con otro *collider*, el mismo que se emplea en el ataque de los enemigos. Al detectar la cercanía de ambos personajes se actualiza con valor verdadero una variable <cerca> de *FruitmanAttack*, de igual manera, cuando estos se alejan, la condición deja de ser cierta. La última variable para controlar los ataques es un temporizador que mide el tiempo entre ataques, análogamente a lo que se hacía con los ataques de los Yokais. Así el usuario no puede atacar continuamente.

4.3. Tercera etapa: Integración con la Kinect

Antes de explicar la integración de *El Desafío de Fruitman* con la Kinect v2, es necesario conocer cómo es el proceso de comunicación de Unity con el middleware K2UM.

La interfaz de Unity para poder comunicarse con el middleware consiste en un *asset* de Unity, *KinectAsset*, que contiene dos *prefabs*: *KinectReceiver y AmplifyManager*. Un *prefab* [24] es un *GameObject* completo e independiente, con sus características y sus componentes, guardado para usarlo repetidas veces en una escena

4.3.1. KinectReceiver

KinectReceiver gestiona la comunicación entre el middleware y Unity, es decir, recibe los datos de las posiciones corporales que capta la Kinect en tiempo real y los aplica a un esqueleto para que haya correspondencia entre los movimientos del usuario y el objeto en Unity. Dicho esqueleto está formado por una serie de *GameObjects* situados en el espacio de tal manera que recuerdan a una figura humanoide y la jerarquía entre ellos respeta la fisionomía humana, es decir, la mano está supeditada al codo, el codo al hombro, etc. En la figura 32 se puede observar que cada una de las articulaciones contempladas se corresponde con una esfera de color amarillo. Sin embargo, comparando con las articulaciones transmitidas por la Kinect, se puede observar que hay cinco articulaciones menos. Las que faltan son las del extremo final debido a que su rotación es nula. La posición de dichas articulaciones es modificable, de tal manera que se puede adaptar a diferentes tipos de esqueletos en función del personaje.

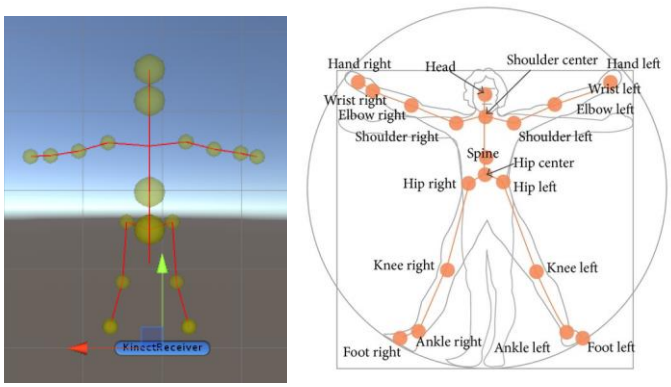

Figura 32. SkeletonReceiver (izqda.) y articulaciones asociadas a la Kinect v2 (dcha.) [24].

Este esqueleto, por tanto, "imita" los movimientos del usuario captados por la Kinect. Se pueden asociar elementos de cualquier *GameObject* de Unity a las articulaciones ya mencionadas gracias al *script Launcher* como se puede ver en la figura 33.

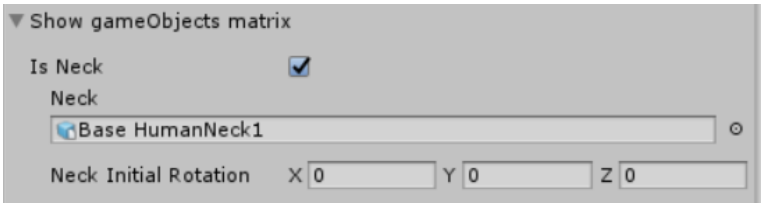

Figura 33. Asociación de una articulación de SkeletonReceiver a un elemento del juego.

De esta manera, el elemento *BaseHumanNeck1* realiza el mismo movimiento que la articulación *Neck*, que a su vez está asociada al movimiento del cuello del usuario que capta la Kinect.

4.3.2. AmplifyManager

Por otro lado, *AmplifyManager* se encarga de amplificar los movimientos del usuario de manera que, si hay un usuario con movilidad limitada, interpreta el máximo desplazamiento de la articulación correspondiente y se lo aplica al esqueleto como si el movimiento se hubiese realizado por completo. Por ejemplo, si el jugador no puede levantar el brazo más de 30 grados respecto de su tronco, se toma este punto como referencia para el desplazamiento máximo que es capaz de realizar el usuario. Por otra parte, se calibra el esqueleto con el movimiento final al que corresponde este desplazamiento, supóngase que el brazo quede en una posición de 90 grados respecto al tronco. De esta manera, cuando el usuario levante el brazo 30 grados, el esqueleto lo levanta 90 y se realiza una interpolación para los valores intermedios. En la figura 34 [], el movimiento realizado por el usuario se representa en el *SkeletonReceiver* (esferas amarillas) y se puede observar que el movimiento final del esqueleto del juego es mayor (el modelo del esqueleto representa un personaje).

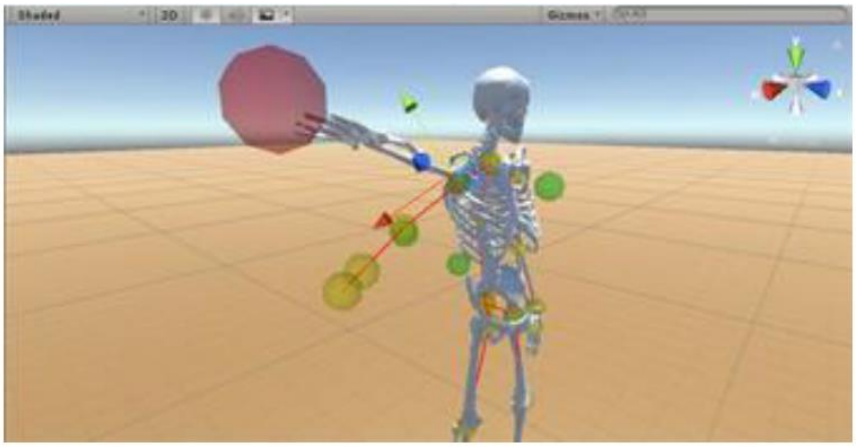

Figura 324. Representación del movimiento amplificado.

4.3.3. Movimiento de Fruitman

Por necesidades del juego, se debe replantear la manera de utilizar el *asset KinectReceiver*, ya que no es posible implementar una relación directa entre el cuerpo de Fruitman y el cuerpo del usuario. Esto se debe a las características del laberinto, que obliga al personaje a moverse en todas las direcciones, mientras que el usuario debe permanecer en un sitio acotado (dentro del rango de captación de la Kinect) y siempre mirando a la pantalla en la que se muestra el juego. Por este motivo, hay que reinventar el uso para el que fue creado inicialmente el middleware.

La solución propuesta consiste en desacoplar esta relación directa entre articulaciones y emplear ciertas partes del *SkeletonReceiver* como controlador del movimiento del usuario. Esto implica no asociar todos los elementos en el inspector de *Launcher*, sino únicamente los necesarios para los ataques: los brazos de Fruitman, como se puede observar en la figura 35.

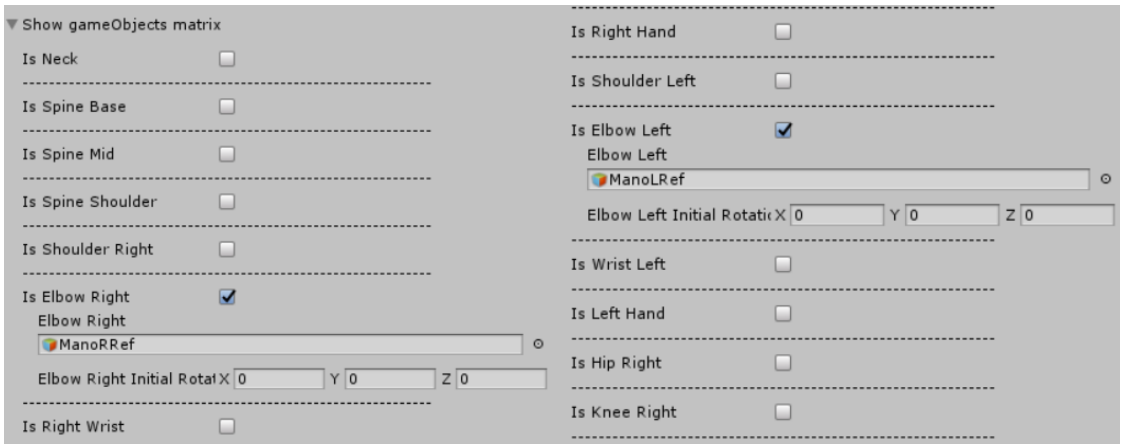

Figura 35. Asociación de articulaciones en el Launcher.

Así, los únicos movimientos del usuario captados por la Kinect que tienen correspondencia directa con los movimientos realizados por Fruitman son los que realizan los codos y se emplean para atacar, ya que es una acción muy intuitiva y que permite una mayor inmersión en el juego. Concretamente se asocian los codos, ya que el movimiento está pensado para la extensión máxima del brazo y es una manera de forzarla: si el codo llega al objetivo, la mano también habrá llegado. Es necesario ajustar el *SkeletonReceiver* para que se adapte a la fisionomía de Fruitman y que los movimientos resulten naturales. La manera de asociar ambos esqueletos viene representada en la figura 36.

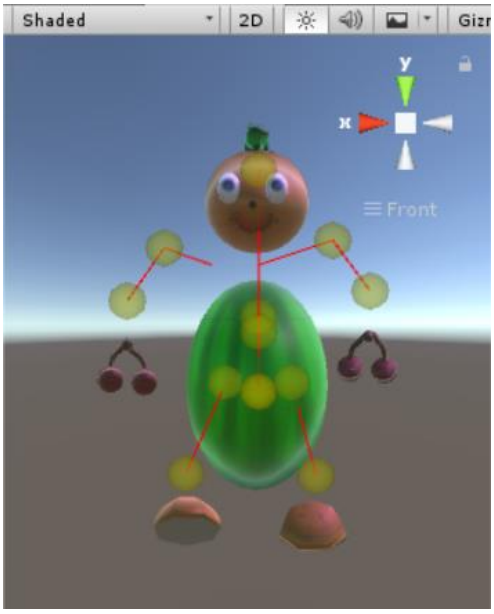

Figura 336. SkeletonReceiver aplicado a Fruitman.

Aunque no se visualicen en el juego todos los movimientos del usuario, se pueden utilizar los datos espaciales de una de las articulaciones para controlar eventos dentro del juego, en este caso, el movimiento de Fruitman. Así, *HipCenterUp*, una de las articulaciones de *SkeletonReceiver* correspondientes a la cadera, tiene como componente un *script* asociado, *Controller*, que gestiona el movimiento de Fruitman. Se elige la cadera porque es un hueso al que los demás están supeditados y por tanto es apto para controlar todo el movimiento del personaje. La posición espacial de este elemento se muestra destacada con una esfera roja en la figura 37. También se puede ver en la figura el *script Controller* que gestiona el movimiento del protagonista.

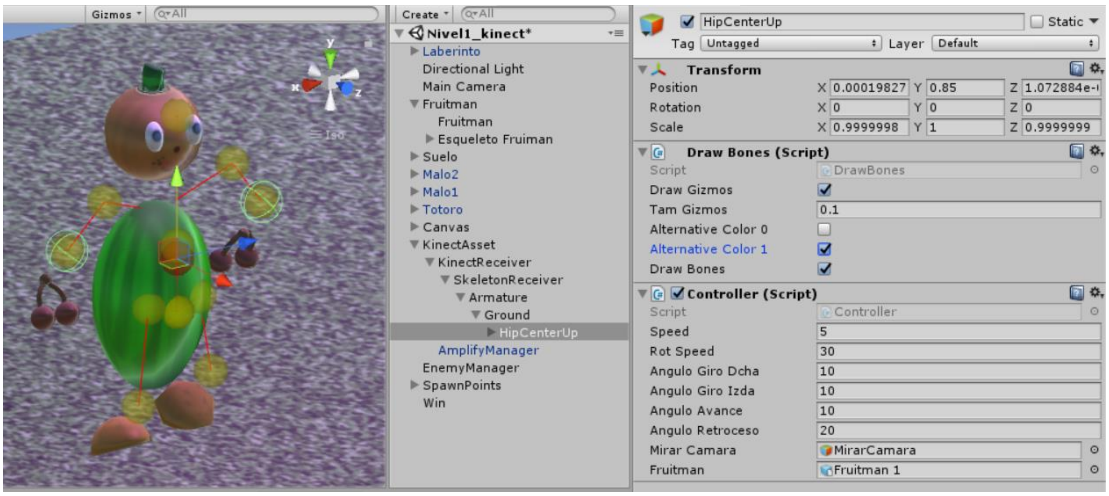

Figura 347. Posición de HipCenterUp.

Es posible acceder a las propiedades de la transformación de este *GameObject* a través del script. De esta manera se pueden obtener los ángulos que forma su transformación respecto a los ejes globales y almacenarlos en variables. Como los movimientos de este

objeto se corresponden con los de la cadera del usuario, se pueden comparar estas variables con unos valores de control y realizar acciones en consecuencia. Así, se puede obtener una funcionalidad añadida de la herramienta *KinectReceiver* al utilizarla como controlador.

La clase *Transform* en Unity tiene una serie de propiedades a las que se puede acceder fácilmente. Mediante *transform.localEulerAngles* es posible obtener el ángulo que forma la transformación de un *GameObject* respecto a un eje en un determinado plano y compararlo con las variables de control. Es decir, si el usuario se inclina hacia delante más de los grados programados, desencadena la acción de avanzar y el cuerpo de Fruitman se mueve en consecuencia. El movimiento se mantiene incluso estando en la posición de reposo. Cuando el usuario quiere detenerse, debe inclinarse hacia atrás más de los grados programados. Visualmente queda indicado en la figura 38.

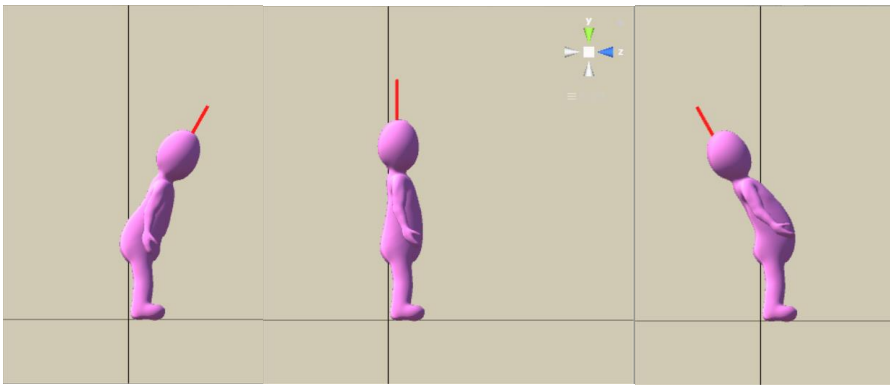

Figura 358. Posiciones de avance, reposo y parada, respectivamente.

Constantemente se compara el ángulo que forma el eje vertical del jugador (línea roja en la figura 38) con el eje Y, y si este es mayor que la variable de control observada en la figura 37, <Angulo Avance>, Fruitman comienza a avanzar a la velocidad <Speed>. En este estado el usuario puede volver a la posición de reposo y el personaje sigue avanzando. Cuando el jugador se inclina hacia atrás un ángulo mayor que <Angulo Stop>, Fruitman deja de avanzar y pasa a estar estático. De manera análoga se implementan los giros a partir del movimiento del tronco superior del usuario y con las variables de control <Angulo Giro Izda> y <Angulo Giro Dcha>, en este caso con el eje X.

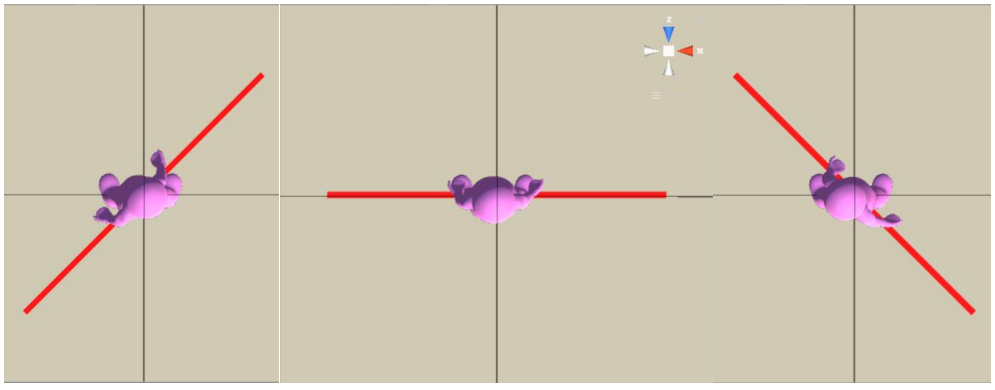

Figura 369. Posiciones de giro a la izquierda, en reposo y a la derecha, respectivamente.

En este punto, una vez explicadas todas las condiciones de las que depende el movimiento, cabe mencionar una serie de variables modificables por el gestor del juego (en este caso los médicos encargados de la evolución de la rehabilitación) que permiten alterar la dificultad del juego. Dicha dificultad radica en los posibles problemas del jugador para realizar movimientos o inclinarse, cuanto más amplio sea el movimiento, mayor dificultad conlleva. Por otra parte, también hay aspectos externos al movimiento del usuario que pueden alterar la dificultad, como hacer que los enemigos ataquen más rápido, quiten más daño o nazcan con mayor frecuencia. Estas variables y su localización dentro del proyecto de Unity son las siguientes:

- AnguloGiroDcha: ángulo en grados a partir del cual el personaje comienza a girar a la derecha.
- AnguloGiroIzda: ángulo en grados a partir del cual el personaje comienza a girar a la izquierda.
- AnguloAvance: ángulo en grados a partir del cual el personaje comienza a avanzar.
- AnguloStop: ángulo en grados a partir del cual el personaje se detiene.

Al pertenecer al movimiento del personaje, estas variables se definen en el *script Controller*, perteneciente a *HipCenterUp,* cuyo *GameObject* padre es *KinectReceiver*.

- TiempoEntreAtaques: determina cada cuánto tiempo atacan los enemigos, en segundos.
- Damage: determina cuánta vida resta a Fruitman cada ataque enemigo (Salud de Fruitman: 100).

Al pertenecer al ataque de los enemigos, estas últimas se definen en el *script EnemyAttack*, asociado a cada uno de los malos (Malo1 y Malo2).

- SpawnTime: determina cada cuántos segundos nace un nuevo malo en una de las posiciones asignadas.

Esta última variable se define en el *script EnemySpawn*, clase que se encarga del nacimiento de los nuevos enemigos y está asociada al *GameObject EnemyManager*.

5. Pruebas de funcionamiento

Durante la realización de este trabajo se han realizado pruebas continuas de depuración de la solución, dando lugar a un proceso de mejora constante. Inicialmente se desarrolló una versión básica del juego controlada por teclado y ratón, para comprobar las interacciones entre los diferentes elementos del juego. En este punto se dio mayor importancia a la lógica del juego, para lo que se evaluaban diferentes criterios, como las colisiones de Fruitman con los enemigos y con el resto de los elementos del juego, el manejo de la cámara o los ataques y el tiempo entre ataques de Fruitman y de los Yokais.

Sin embargo, al ser un videojuego controlado a través de los movimientos corporales, las pruebas más interesantes se realizaron una vez estaba implementada la conexión con la Kinect v2. Las pruebas fueron realizadas por cuatro sujetos tanto de pie como sentados, ya que el videojuego debe ser apto para personas en silla de ruedas. Las características físicas de estos sujetos se muestran en la tabla 1.

Tabla 1. Características de los sujetos de prueba.

	Edad	**Sexo**	**Altura**	**Peso**	**Condición física**
Sujeto 1	24	M	177	74 kg	Buena. Realiza deporte regularmente.
Sujeto 2	21	F	169	63 kg	Normal.
Sujeto 3	23	M	176	83 kg	Buena. Realiza deporte ocasionalmente.
Sujeto 4	67	M	182	85 kg	Regular. Refiere dolores de espalda que impiden movimientos bruscos.

Las primeras pruebas de funcionalidad ya daban lugar a realizar un cambio en la forma de controlar el personaje de Fruitman. Inicialmente, para que Fruitman avanzara o retrocediera, el usuario tenía que inclinarse más que los grados programados y mantenerse en esa posición. Si volvía a la posición de reposo, Fruitman se detenía. Para cambiar la dirección de avance, el usuario tenía que girar el tronco. Esta lógica de movimientos no gustó a los sujetos e incluso impedía la correcta finalización del juego. Hubo varios problemas referidos unánimemente durante las pruebas: primero, la postura de inclinación del cuerpo había que mantenerla durante prácticamente todo el juego, lo que es molesto físicamente e impedía ver bien la pantalla, ya que había que echar la cabeza hacia atrás. Además, era muy complicado realizar un avance con giro ya que implicaba la realización de dos movimientos simultáneos: inclinarse y girar. A parte de ser incomodo, puede ser contraindicado para personas con ciertas dificultades o restricciones físicas. Por estos motivos se decidió que Fruitman debía avanzar automáticamente y que el usuario únicamente usa los movimientos de inclinación hacia adelante y hacia atrás para empezar a andar y frenarle, lo cual además son movimientos más naturales.

Por otra parte, las pruebas revelaban incomodidad para realizar los movimientos de ataque. En el desarrollo inicial se requería un movimiento similar al de un puñetazo, es decir, extender el brazo de atrás hacia adelante perpendicular al pecho. Ya las primeras pruebas descartaron esta idea debido a la poca precisión en la captación de movimiento. Al ser un movimiento del brazo en dirección a la cámara, entra en juego la profundidad y la resolución del hardware en este aspecto no es tan alta. Debido a este problema se decidió que el movimiento de ataque consistiese en subir los brazos. Este modelo tampoco convencía porque resultaba antinatural realizar un ataque de esa manera, no hay fuerza en ese movimiento. Finalmente, se optó por bajar los brazos, que tenía mayor fuerza e impacto de cara a la inmersión en el juego.

Realizados estos cambios, las últimas pruebas dieron resultados satisfactorios, que quedan reflejados en la tabla 2.

Tabla 2. Resultados de las sesiones de prueba.

Sujeto 1	**Éxito**	**Tiempo**	**Comentarios**
Sesión 1	No	2:35	Tarda en encontrar la salida del laberinto y acaba perdiendo la salud
Sesión 2	No	3:10	Tarda en encontrar la salida y acaba perdiendo la salud
Sesión 3	Sí	3:11	-
Sesión 4	Sí	2:53	-

Sujeto 2	**Éxito**	**Tiempo**	**Comentarios**
Sesión 1	No	1:48	Problemas al eliminar los enemigos
Sesión 2	No	2:57	Tarda en encontrar la salida y acaba perdiendo la salud
Sesión 3	Sí	3:57	-
Sesión 4	Sí	3:21	-

Sujeto 3	**Éxito**	**Tiempo**	**Comentarios**
Sesión 1	No	2:50	Tarda en encontrar la salida y acaba perdiendo la salud
Sesión 2	Sí	3:23	Incomodidad con la cámara cerca de las paredes
Sesión 3	Sí	2:45	-
Sesión 4	Sí	3:21	-

Sujeto 4	**Éxito**	**Tiempo**	**Comentarios**
Sesión 1	No	1:15	Problemas para controlar el juego, falta de entendimiento
Sesión 2	No	2:13	Mejor entendimiento, pero eliminado por enemigos
Sesión 3	No	2:55	Tarda en encontrar la salida y acaba perdiendo la salud
Sesión 4	Sí	3:33	Consigue el éxito, ayudado con indicaciones

Se puede observar en estos resultados que ninguno de los sujetos pudo pasarse *El Desafío de Fruitman* la primera vez, pero tras varios intentos todos fueron capaces de jugar al juego exitosamente. Los primeros intentos son normalmente frustrados por falta de práctica y desconocimiento del juego y el entorno, sin embargo, los jugadores adquirieron rápidamente la destreza necesaria para desenvolverse sin problemas. El caso más problemático es el del sujeto 4, pero los malos resultados se deben también a su avanzada edad y la falta de práctica en jugar a videojuegos.

6. Conclusiones

El objetivo principal de este trabajo era incorporar al proyecto Blexer un juego en tres dimensiones utilizando el software Unity y que fuese controlable por la interfaz de usuario natural proporcionada gracias al middleware K2UM y a la cámara Kinect v2. Este proyecto planteaba ciertos desafíos adicionales tales como partir de un videojuego ya creado para Blender, *Buscando a Totoro*. Por ello, se ha desgranado en tres bloques de trabajo: exportación en Blender e importación en Unity, generación de la lógica en Unity basada en scripts C# e integración con la Kinect v2 para la captación de movimiento.

En la etapa de exportación de Blender e importación a Unity surgieron multitud de problemas, sobre los que hubo que investigar las causas raíces y solucionarlas. Muchos de los problemas venían provocados por una jerarquía ineficiente de los objetos en Blender o por malas posiciones relativas de los elementos respecto al origen. Estos problemas fueron subsanados, cumpliendo uno de los objetivos principales.

Los modelos de Blender se han aprovechado y se utilizan para no perder la esencia inicial del videojuego, sin embargo, la lógica se ha construido desde cero, a partir de diferentes *scripts* que dotan de coherencia, jugabilidad y fluidez al juego.

Para cumplir el tercer objetivo se partía del middleware K2UM ya desarrollado, que aunque inicialmente no estuviese optimizado el uso que se le da en este trabajo, se halló una solución alternativa que permitía al usuario controlar al protagonista del juego con sus propios movimientos corporales. Esta nueva funcionalidad admite una mayor versatilidad a la hora de controlar eventos en el juego, ya que cada movimiento puede desencadenar una acción, como si de un mando se tratase.

Para formar parte de Blexer es necesario crear un módulo que permita la interconexión con la plataforma Blexer-Med. Dicho módulo debe enviar y recibir datos del juego para que un especialista médico pueda monitorizar los avances del paciente y gestionar la dificultad de los ejercicios. Esta condición se ha tenido en cuenta desde las fases iniciales del desarrollo y se han definido unas variables o parámetros, mencionadas en el apartado de pruebas, que permiten variar fácilmente la dificultad del videojuego.

El reto del proyecto consistía en desarrollar un videojuego manejable a través de una interfaz natural de usuario gracias al hardware Kinect. Por falta de tiempo finalmente se tomó la decisión de realizar un único nivel en *El Desafío de Fruitman*, lo que varía ligeramente la modalidad de juego respecto al videojuego original. En esta versión no hay un cronómetro contrarreloj para salir del laberinto, pero la esencia del juego y el modus operandi son los mismos: rescatar a Fruitman y escapar juntos del laberinto. A pesar de los problemas y limitaciones técnicas, se han encontrado soluciones viables y se ha conseguido sacar el proyecto adelante. Este documento saca a relucir posibles problemas que se pueden encontrar en el desarrollo de juegos del estilo e incluso en problemas de interoperabilidad entre diferentes programas involucrados en el proceso de creación de un videojuego. El resultado final es un juego íntegro que permite ser manejado a través de una interfaz natural

de usuario y se enmarca dentro del proyecto Blexer, en el que se siguen desarrollando juegos para la rehabilitación física.

7. Futuras líneas de trabajo

El Desafío de Fruitman es un videojuego con mucho potencial por desarrollar. Actualmente está desarrollado como un solo nivel con un objetivo: llegar al centro del laberinto, encontrar a Totoro y salir juntos. Hay varias maneras relativamente sencillas de aumentar la jugabilidad sin perder la esencia del juego.

Para mejorar la inmersión en el juego se podría mejorar la perspectiva de la cámara, porque actualmente hay ciertos puntos ciegos, donde las paredes son muy estrechas, en los que la visibilidad es complicada. Esto se puede solucionar mejorando el *script* asociado a la cámara de tal manera que se cambie a primera persona cuando el espacio es reducido. También se podrían introducir ciertos elementos estéticos como un flash rojo intermitente que pusiese en tensión al jugador cuando a Fruitman le queda poca vida. Otro elemento que puede incrementar la intensidad es añadir una cuenta atrás configurable en el momento en que Fruitman encuentra a Totoro.

Otro punto por mejorar es la duración del juego. Las líneas de trabajo en esta dirección podrían ser:

- Concebir el juego como una serie de ejecuciones recursivas en las que se vaya incrementando progresivamente la dificultad gracias a las variables configurables. Se podría llevar un registro de la puntuación máxima con nombres como las máquinas de *arcade* antiguas.
- Introducir algún tipo de consumible que permita recuperar salud. Esto toma importancia a medida que la dificultad aumenta.
- Para hacer el juego menos monótono se pueden modificar las características físicas de los Yokais enemigos, como el tamaño, la velocidad o la salud que quitan con sus ataques.

Para mejorar la estética y la inmersión, sería conveniente crear una pequeña cinemática que introduzca al jugador en el contexto del juego. Un posible argumento es mostrar a Totoro siendo capturado por los Yokais y posteriormente a Fruitman, cuando se entera de que su amigo está en peligro, con algún texto que implique determinación y la voluntad de ayudarlo. Puede desarrollarse en los exteriores del laberinto para situar directamente el contexto geográfico.

Otro punto clave abierto a mejora es la comunicación con la página web que permite a los terapeutas monitorizar y gestionar el progreso de sus pacientes y aún no está implementada. Para ello sería necesario crear un módulo dentro del juego que pudiese modificar las variables mencionadas en el apartado de pruebas y enviar los resultados de las sesiones de trabajo para la posterior evaluación por el especialista.

El proyecto Blexer podría incorporar más interfaces naturales de usuario existentes en el mercado: el mando de la consola Wii, que incorpora acelerómetros que permiten medir el movimiento en los tres ejes; la Wii Balance Board, un accesorio en forma de tabla capaz

de medir la presión ejercida sobre ella y que sería muy útil en ejercicios de equilibrio; sistemas de realidad virtual como Oculus Rift, o incluso cualquier smartphone o Tablet, ya que llevan incorporados acelerómetros que pueden detectar los movimientos en mayor o menor precisión. Incorporar estos hardware diferentes puede tener un coste elevado en cuanto a recursos, aunque la historia del proyecto Blexer demuestra que ya se ha sido capaz de cambiar de hardware exitosamente.

8. Referencias

[1] Pamela M. Kato; "Video Games in Health Care: Closing the Gap". Review of General Psychology, American Psychological Association 2010, Vol. 14, No. 2, pp. 113–121 DOI: 10.1037/a0019441113. Disponible en:

https://studylib.net/doc/18833632/video-games-in-health-care--closing-the-gap

[2] Judith E Deutsch, Megan Borbely, Jenny Filler, Karen Huhn, Phyllis Guarrera-Bowlby; Use of a Low-Cost, "Commercially Available Gaming Console (Wii) for Rehabilitation of an Adolescent With Cerebral Palsy", *Physical Therapy*, Volume 88, Issue 10, 1 October 2008, Pages 1196–1207, Disponible en: https://doi.org/10.2522/ptj.20080062

[3] Griffin M.; Shawis T.; Impson R.; McCormick D., Taylor M. D. (2012). "Using the Nintendo Wii as an Intervention in a Falls Prevention Group". Journal of the American Geriatrics Society. 60 (2): 385–387. doi:10.1111/j.1532-5415.2011.03803.x

[4] UPM "Grupo de Aplicaciones Multimedia y Acústica (GAMMA)". [última consulta 18 noviembre 2018]. Disponible en:

https://www.citsem.upm.es/index.php/es/personal/grupos/personal-gamma

[5] UPM "Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM)". [última consulta 18 noviembre 2018]. Disponible en:

https://www.citsem.upm.es

[6] Gómez-Martinho, I., "Desarrollo e implementación de middleware entre Blender, Kinect y otros dispositivos", Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, Julio 2016.

[7] Luaces Vela, C., "Diseño e implementación de un entorno virtual de ejercicios físicos, basados en captura de movimiento", Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, Mayo 2018.

[8] Eckert, M., Gómez-Martinho, I., Esteban, C., Peláez, Y., Meneses, J. y Salgado, L., "Blexer – Full Play Therapeutic Blender Exergames for People with Physical Impairments." 3rd EAI International Conference on Smart Objects and Technologies for Social Good, Pisa, Italy, Nov 29–30, 2017.

[9] Jiménez, M., "Plataforma médica para el entorno del videojuego terapéutico *BLEXER*", Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, Julio 2017.

[10] Eckert, M., Jiménez, M., Martín-Ruiz, M-L., Meneses, J. and Salgado, L., "Blexer-med – A medical web platform for administrating full play therapeutic Exergames." 3rd EAI International Conference on Smart Objects and Technologies for Social Good, Pisa, Italy, Nov 29–30,2017.

[11] Eckert, M., Gómez-Martinho, I., Meneses, J., Martínez, J.F., "New Approaches to Exciting Exergame-Experiences for People with Motor Function Impairments," Sensors 17(2), 2017.

[12] Luaces, C., "Diseño e implementación de un entorno virtual de ejercicios físicos, basados en captura de movimiento" Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, Mayo 2018.

[13] Corporales, C., López, P., Zambrano, A., "Game Design Document: Buscando a Totoro" Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, Junio 2017.

[14] Ubi Soft Entertaining. *Rayman 2: The Great Escape Manual*. Reino Unido: Ubi Soft Entertaining, 1999.

[15] Rayman – Origins Anniversary Art Jam, Alexandra Amerin. ©2016-2019 EarthGwee [última consulta 18 noviembre 2018]. Disponible en:

https://www.deviantart.com/earthgwee/art/Rayman-Origins-Anniversary-Art-Jam-643536893

[16] Totoro. Disney Fandom. 2013. [última consulta 18 noviembre 2018]. Disponible en: http://disney.wikia.com/wiki/Totoro

[17] Unity. © 2019 Unity Technologies. [última consulta 18 noviembre 2018]. Disponible en: https://unity3d.com/

[18] Blender Project. [consulta 18 noviembre 2018]. Disponible en: https://www.blender.org/

[19] FBX. Wikipedia: the free encyclopedia. 27 febrero 2018. [última consulta 18 noviembre 2018]. Disponible en: https://en.wikipedia.org/wiki/FBX

[20] What is Autodesk FBX technology? © Autodesk Creative Commons. [última consulta 18 noviembre]. Disponible en:

http://help.autodesk.com/view/FBX/2017/ENU/?guid=_files_GUID_274163DA_9E89_4DCC_8AF6_10B0C498582E_htm

[21] Unity User Manual. © 2019 Unity Technologies. [última consulta 18 noviembre 2018]. Disponible en: https://docs.unity3d.com/Manual/index.html

[22] Understanding Kinect V2 Joints and Coordinate System. Lisa Jamhoury. 23 julio 2018. [última consulta 18 noviembre 2018]. Disponible en:

https://medium.com/@lisajamhoury/understanding-kinect-v2-joints-and-coordinate-system-4f4b90b9df16

[23] Blender Manual: NLA Editor. [última consulta 18 noviembre 2018]. Disponible en: https://docs.blender.org/manual/de/dev/editors/nla/introduction.html

[24] Hand Gesture and Character Recognition Based on Kinect Sensor - Scientific Figure on ResearchGate. [última consulta 8 febrero 2019]. Disponible en:

https://www.researchgate.net/figure/Recognition-of-the-human-skeleton-from-Microsoft-Kinect-for-Windows-SDK_fig8_275470661

Anexo 1. Presupuesto.

A continuación, se detallan brevemente los detalles económicos de este proyecto. El equipamiento necesario para realizarlo es un ordenador con capacidad para correr Unity 3D y el hardware Kinect v2. Por otra parte, la mano de obra asciende a un total de 300 horas de ingeniería. Todo esto queda reflejado en la siguiente tabla:

Tabla 3. Presupuesto del proyecto.

Concepto	Unidades	Coste Unitario	Total
Ordenador Portátil	1	800	800
Kinect v2	1	65	65
Horas de ingeniería	300	20	6000

Coste total del proyecto	**6865**

Es decir, el coste total del proyecto asciende a un monto de 6865 euros, a lo que añadiendo el 21% del IVA queda en **8306,65** euros.

Anexo 2. Manual del juego.

En la siguiente página se presenta un manual del videojuego.

EL DESAFIO DE FRUITMAN

Meta

¡Oh, no! ¡Los malvados Yokais han secuestrado a Totoro!
En esta aventura tendrás que controlar al valiente Fruitman para rescatar a su mascota favorita.

Intérnate en la laberíntica guarida de estos demonios de los bosques y encuentra a tu amigo.

¡Cuidado! Cuando lo encuentres los enemigos ya habrán dado la voz de alarma y tendréis que salir juntos pitando de ahí o si no… ¡acabaréis capturados otra vez!

¡Corre, Totoro está esperando su rescate!

Controles

Avanzar -> Inclinarse hacia adelante

Detenerse -> Inclinarse hacia atrááas

Girar -> Girarse hacia los lados

Atacar -> bajar los brazos

Créditos

Pablo López Miedes

Basado en “Buscando a Totoro”, de Pablo López Miedes, Cristina Corporales Morente y Angélica Zambrano Suárez.