PROYECTO FIN DE GRADO

TÍTULO: Diseño y creación de un videojuego terapéutico como base para su posterior ampliación con algoritmos inteligentes.

AUTOR/A: Sandra Ceballos Manso

TITULACIÓN: Ingeniería de Sonido e Imagen

TUTOR/A: Martina Eckert

DEPARTAMENTO: Ingeniería Audiovisual y Comunicaciones

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Marta Sánchez Agudo

TUTOR/A: Martina Eckert

SECRETARIO/A: Enrique Rendón Angulo

Fecha de lectura: 22/07/2024

Calificación:

El Secretario/La Secretaria,

Resumen

Este proyecto de fin de grado tiene como principal objetivo el diseño y creación de un videojuego terapéutico como base para su posterior ampliación con algoritmos inteligentes. Esto surge de la necesidad de cambio en el trascurso de los juegos para despertar el interés de los pacientes y hacer de la terapia algo motivador y gratificante, obteniendo unos mejores resultados.

Para llevar a cabo este proyecto, se ha estudiado el funcionamiento del anterior juego ya existente en el grupo de investigación "*Phiby's* Adventures 3D" y en base a este se ha realizado uno a menor escala, con unos objetivos propuestos y mejorando ciertos problemas que se presentaban a la hora de jugarlo.

Las características implementadas en este nuevo juego consisten en: un entorno no navegable en el que el recorrido del personaje sea configurable a través de código; un sistema de minijuegos estructurado en escenas independientes y que mantengan los movimientos del juego en el que se basa; capacidad de adaptación a cada usuario a través de la conexión con una plataforma web; una estructura fácil de modificar y, por último, un carácter atractivo y motivador para el paciente.

El proyecto se ha desarrollado utilizando la plataforma *Unity* para la creación de videojuegos y el editor de código *Visual Studio 2019* para la creación de código con el lenguaje C#. En cuanto a la captación de movimientos, se ha utilizado la cámara Kinect v2 de Microsoft.

Económicamente se ha considerado la utilización de estas herramientas porque son relativamente asequibles, lo que facilita que el sistema pueda ser implementado en hogares y centros de rehabilitación. Por otro lado, desde un punto de vista ambiental, el uso de tecnología digital para la rehabilitación contribuye a reducir el consumo de recursos físicos que de otra manera serían necesarios para terapias tradicionales.

Como resultado se ha logrado desarrollar un videojuego funcional que incluye los principales objetivos propuestos además de otras funcionalidades. Tras realización de las pruebas iniciales se sostiene que el juego contiene todo lo propuesto y, además de esto y para futuras modificaciones, se han propuesto líneas de trabajo a través de las cuales es posible extender el juego y aplicar estos algoritmos inteligentes mencionados.

Abstract

The main objective of this final degree project is the design and creation of a therapeutic videogame as a basis for its subsequent extension with intelligent algorithms. This arises from the need for change in the course of the games to arouse the interest of patients and make the therapy something motivating and rewarding, obtaining better results.

To carry out this project, the operation of the previous game already existing in the research group "*Phiby's Adventures 3D*" was studied and based on this one was made on a smaller scale, with some proposed objectives and improving certain problems that appeared when playing it.

The features implemented in this new game consist of: a non-navigable environment in which the character's path is configurable through code; a system of mini-games structured in independent scenes that maintain the movements of the game on which it is based; the ability to adapt to each user through the connection with a web platform; a structure easy to modify and finally, an attractive and motivating character for the patient.

The project has been developed using the *Unity* platform for the creation of video games and the *Visual Studio 2019* code editor for the creation of code with the C# language. As for motion capture, the Microsoft Kinect v2 camera has been used.

Economically, the use of these tools has been considered because they are relatively affordable, which makes it easier for the system to be implemented in homes and rehabilitation centers. On the other hand, from an environmental point of view, the use of digital technology for rehabilitation contributes to reduce the usage of physical resources that would otherwise be necessary for traditional therapies.

As a result, it has been possible to develop a functional videogame that includes the main proposed objectives as well as other functionalities. After carrying out initial tests, the game contains everything proposed and, in addition to this and for future modifications, lines of work have been proposed through which it is possible to extend the game and apply these intelligent algorithms mentioned.

Índice de figuras

Índice de tablas

Lista de acrónimos

Blexer:	Blender Exergames
CITSEM:	Centro de Investigación de Tecnologías Software y Sistemas Multimedia para la Sostenibilidad
GAMMA:	Grupo de Aplicaciones Multimedia y Acústica
IA:	Inteligencia Artificial
K2UM:	Kinect to Unity Middleware
NPC:	Non Playable Character

Índice de contenidos

1. Introducción

1.1 Marco y motivación del proyecto

La unión entre tecnología y salud en el mundo actual está abriendo nuevas fronteras en el tratamiento y recuperación de diversas afecciones médicas. Una de las innovaciones más prometedoras en este campo es el uso de videojuegos terapéuticos, que ofrecen soluciones que combinan el entretenimiento con efectos terapéuticos. Es por esto por lo que organizaciones como GAMMA (Grupo de Investigación de Aplicaciones Multi-Media y Acústica) de la Universidad Politécnica de Madrid dentro del CITSEM (Centro de investigación de Tecnologías Software y Sistemas Multimedia para la sostenibilidad) inició una línea de trabajo sobre el diseño y desarrollo de videojuegos serios para personas con problemas de movilidad.

Uno de los proyectos en funcionamiento de este grupo es el sistema *BLEXER* (*Blender Exergames*), en desarrollo desde 2015. Está basado en un entorno de *exergaming* que se adapta al usuario para hacer ejercicios personalizados y que está conectado con una página web con base de datos (*Blexer-med*), a la cual el terapeuta accede. Para la captura de movimientos realizados por el usuario, se utiliza la cámara *Kinect v2* de *Microsoft* la cual, a través del middleware K2UM, envía la información de cada punto del cuerpo del jugador al juego. Además de este traspaso de información, el middleware también realiza la comunicación entre la web y el juego.

Dentro del grupo y de este entorno se encuentra el juego "*Phiby's Adventures 3D*", en el cual está basado este proyecto. Se trata de un entorno explorable en donde un personaje debe ir avanzando en la historia y donde se suceden una serie de minijuegos que el usuario debe completar realizando movimientos con los brazos y tronco. Es de este juego que se han reutilizado el personaje y los movimientos principales para los minijuegos.

1.2 Objetivos técnicos y académicos

Este proyecto de fin de grado, "*Phiby's Toyland Escape*" tiene como objetivo principal la creación de un videojuego terapéutico destinado a jóvenes con movilidad reducida, específicamente entre las edades de 7 y 14 años. Este juego no solo busca ser una herramienta terapéutica, además debe servir como base para que en el futuro se implante inteligencia artificial para personalizar la experiencia del usuario.

El diseño de este debe tener una estructura bien definida y modular, facilitando así la incorporación de futuras mejoras y adaptaciones por parte de estos algoritmos inteligentes.

Además, se debe garantizar que el videojuego es accesible para jóvenes con diferentes niveles de movilidad reducida, asegurando una experiencia de usuario inclusiva y satisfactoria.

1.3 Estructura del resto de la memoria

En la presente memoria se aborda el desarrollo de un videojuego terapéutico en Unity, diseñado para personas con problemas de movilidad. Este proyecto tiene como objetivo proporcionar una herramienta lúdica y efectiva para la rehabilitación física, además de sentar las bases para futuras integraciones de inteligencia artificial.

El capítulo dedicado al marco tecnológico revisa la literatura existente sobre videojuegos terapéuticos y su evolución reciente. Se analizan diferentes herramientas y tecnologías disponibles en el mercado y antecedentes en los que se basa este proyecto. Además, se visualiza el uso de inteligencia artificial en videojuegos, subrayando su potencial para personalizar y mejorar la experiencia de los usuarios.

Tras detallar las especificaciones y restricciones que guían el desarrollo del proyecto, se profundiza en la implementación técnica en Unity y a través de código en el apartado de descripción de la solución propuesta, junto con el análisis del concepto y narrativa del juego. También se describe la integración con el middleware K2UM, lo que permite la conexión con la plataforma *"Blexer-med*" para el seguimiento y la personalización de los ejercicios.

En resultados, se detallan las pruebas realizadas para asegurar que el videojuego cumple con los objetivos propuestos y se presentan los resultados obtenidos.

Finalmente, en conclusiones, se resume el logro de los objetivos del proyecto y se reflexiona sobre su impacto en el campo de la rehabilitación y los videojuegos terapéuticos en el apartado de impacto del proyecto. Se plantean los trabajos futuros, centrándose en integración de inteligencia artificial para mejorar aún más la efectividad y el alcance de estos videojuegos.

En referencias, se enumeran todas las fuentes utilizadas a lo largo del proyecto, y en los anexos se incluye documentación técnica adicional, como códigos fuente y diagramas de flujo. Éstos complementan y amplían la información presentada en los capítulos principales, además del manual de usuario y el presupuesto que se ha manejado para el proyecto.

2. Marco tecnológico

Los videojuegos terapéuticos son una nueva forma de ayudar a personas con ciertas patologías, a recuperarse a partir de la unión de la tecnología y la medicina. En concreto, en los últimos años, el uso de videojuegos en la salud de niños, especialmente con movilidad limitada, ha ganado gran reconocimiento, pasando de ser un mero entretenimiento para servir de gran ayuda.

Cuando se trata de rehabilitación, es bien sabido que la actitud y el compromiso del paciente son esenciales para su progreso. Sin embargo, para muchos niños, participar en actividades de rehabilitación puede resultar aburrido y desmotivador. Esto puede afectar negativamente su disposición a participar y, por tanto, su recuperación. Sin embargo, cuando estas actividades se presentan de manera lúdica e interactiva, como en ciertos juegos diseñados para la recuperación, esto despierta la curiosidad del niño y fomenta un mayor interés en participar en ellas. Este enfoque no sólo cambia la percepción de la rehabilitación, haciéndola más atractiva y gratificante para los niños, sino que también puede mejorar su disposición a continuar con el tratamiento y con esto, sus resultados terapéuticos.

Para seguir con el tema del entretenimiento, los videojuegos a veces pueden carecer de variedad y resultar algo lineales, lo que limita su capacidad para mantener el interés del jugador a lo largo del tiempo. Es por esto por lo que ha surgido un interés creciente en la integración de inteligencia artificial (IA) en los videojuegos con el propósito de añadir un complemento adicional de complejidad y dinamismo, permitiendo una evolución menos predecible. Introduciendo estos elementos a los juegos, entra la posibilidad de adaptar cada sesión a las necesidades y capacidades específicas de cada individuo, además de aumentar, como ya se ha mencionado, la motivación del paciente y optimizar los resultados terapéuticos.

Además, otro de los factores que se está incorporando, es la utilización de realidad virtual en juegos, con la intención de simular actividades cotidianas en entornos virtuales controlados. Esto se presenta como una herramienta prometedora para ampliar los límites de la rehabilitación proporcionando nuevas formas de mejorar la movilidad y calidad de vida de las personas con limitaciones físicas.

Por otro lado, encontramos el inconveniente de accesibilidad que presentan las personas con algún tipo de incapacidad física. Con accesibilidad nos referimos a «la capacidad de jugar a un juego incluso cuando se funciona en condiciones limitantes. Las condiciones limitantes pueden ser limitaciones funcionales o discapacidades, como la ceguera, la sordera o las limitaciones de movilidad». Esto está definido por el *Game Accessibility Special Interest Group* (GS-SIG) de la International *Game Developers Association* (IGDA) [1], la cual pretende crear juegos universalmente accesibles para todos sin la influencia de edad, experiencia o capacidades. Aunque haya juegos específicamente para colectivos con diversidad funcional, a largo plazo el objetivo debería ser avanzar hacia videojuegos comerciales lo más universalmente accesible posible [2].

2.1 Estado del arte

Las investigaciones ante la utilización de videojuegos para terapias se iniciaron en 1996 con las primeras experiencias de Hunter Hoffman y David Patterson, presentando resultados positivos de la aplicación de realidad virtual en el tratamiento del dolor [3]. A partir de este comienzo, se siguió avanzando con investigaciones en el campo, siendo de las más significativas las pruebas realizadas en el Hospital Universitario de Newark. Aquí, se demostró en una serie de niños que jugaron a videojuegos de la *GameBoy* durante momentos antes de la intervención, que mostraban menor nivel de ansiedad que los que optaron por no jugar [4].

Años después, la organización *HopeLab* realizó un estudio con pacientes diagnosticados con cáncer entre 13 y 29 años para lo que crearon el videojuego *RE-Mission* [5]. Los resultados mostraron que el 80 % de los jóvenes que lo usaron, siguieron el tratamiento con mayor interés y conocieron su enfermedad más traduciéndose esto en una mayor aceptación del cáncer [4].

Los beneficios de estas técnicas, como una mejor coordinación, flexibilidad corporal y tiempos de reacción más rápidos se confirman por estos estudios y muchos más. Esto respalda la efectividad de los videojuegos en la rehabilitación física, mostrando cómo pueden mejorar una variedad de habilidades físicas y mentales.

A medida que surgieron estos nuevos métodos y se visualizaron resultados favorables, diferentes instituciones de investigación y laboratorios comenzaron a desarrollar videojuegos para explotar estas técnicas. Uno de los materiales más recientes es el desarrollado por la Universidad Nacional Autónoma de México (UNAM) a través del Laboratorio de Investigación y Desarrollo de Aplicaciones Interactivas en Neurorrehabilitación (LANR) [6]. Este, cuenta con 12 videojuegos registrados, de los cuales cuatro están disponibles al consumidor, destacando "*Penal Madness"* o "*Sandwichmania"* [7].

Estos softwares surgen a partir de los lanzados por las principales plataformas de videojuegos como son *Xbox Kinect* de *Microsoft*, Realidad Virtual o *Nintendo Wii*. Destacando títulos como "*Wii Fit*" (Nintendo) [8] o "*Kinect Sports*" desarrollado para *Xbox*, siendo este último el principal influyente del proyecto actual, ya que basa el juego en la captura de movimientos a través de una cámara Kinect. Estos dos, establecen su lógica en una serie de deportes que deben ser realizados para promover el movimiento y la actividad física.

Otro ejemplo que podemos encontrar en el mercado es MIRA Rehab [9], un software que permite jugar a diversos videojuegos clínicamente construidos y adaptados al plan de fisioterapia del paciente. Se puede utilizar tanto en la clínica como en casa, lo que permite la monitorización remota de los pacientes y ejercicios más precisos para tiempos de recuperación más rápidos.

Ante esto, el grupo de investigación GAMMA perteneciente al CITSEM, comenzó una nueva línea de investigación sobre el estudio de interfaces naturales inteligentes para personas con discapacidad física [10], centrándose en la realización de ejercicios de mantenimiento a través de la gamificación. Para este proyecto se va a centrar el foco en la línea de *Blender Exergames*.

2.2 Trabajo previo en el que se basa

2.2.1 El entorno Blexer

El entorno Blexer (*Blender Exergames*), es un sistema de ejercicios físicos dirigido a personas con diversidad funcional. El propósito de este proyecto es desarrollar videojuegos que empleen las cámaras *Kinect One* para registrar los movimientos del usuario y que la dificultad de los ejercicios sea ajustable a través de internet. Así, los terapeutas especializados pueden personalizar los ejercicios para la rehabilitación de cada paciente. En el diagrama de flujo de la Figura 1 pueden verse las comunicaciones que se producen en el sistema Blexer.

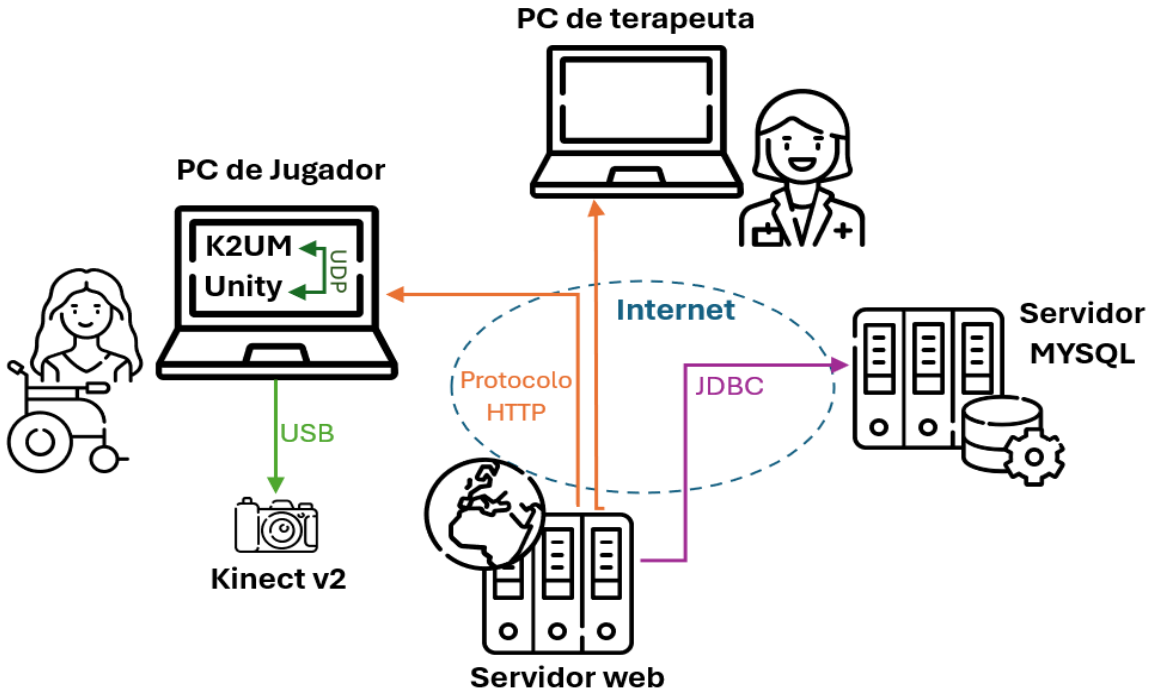

Figura 1.Diagrama de comunicación de los elementos del sistema Blexer.

Para la comunicación de los terapeutas con los juegos, el entorno cuenta con *Blexer-med* [11], que consiste en un servidor web y un servidor MYSQL. El sistema fue creado por Mónica Jiménez [12] y mejorado por Alba Aguilar [13] con el fin de ajustar la dificultad de los ejercicios para personalizar los juegos y mantener el registro de los resultados y acotaciones de los pacientes en cada uno de los videojuegos del sistema. El proceso de descarga y envío de datos será analizado en puntos posteriores.

De esta forma, los terapeutas determinan los diferentes parámetros de cada sesión de los juegos y visualizan los resultados de las sesiones ya finalizadas. Esto permite llevar un seguimiento exhaustivo de la terapia y la adaptación a las necesidades de cada jugador (se explica más detalladamente en el apartado 4.7).

Para la comunicación entre el videojuego y la web, César Luaces [14] creó el Middleware *K2UM* (*Kinect to Unity Middleware*) que se ejecuta en el ordenador del paciente en paralelo al juego. Originalmente fue diseñado para la transmisión de los datos de movimiento, obtenidos por la cámara al juego, pero ha ido evolucionando para actuar también como puente entre el videojuego y el servidor web. En este proyecto se ha usado en su última versión, realizada por Filip Racki [15], la cual integra la posibilidad de utilizar dos tipos de cámaras, la *Kinect* v2 para *Xbox One* y la cámara *Kinect Azure*, ambas creadas por *Microsoft*.

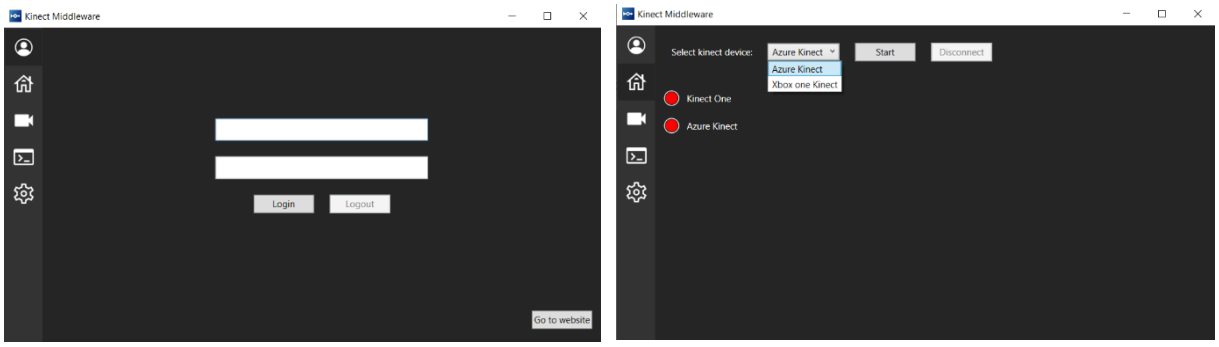

Figura 2. Interfaz de usuario del middlewar K2UM (versión de Filip Racki).

La interfaz del middleware puede visualizarse en la figura 2.

Para el sistema se crearon ya una multitud de videojuegos con fines parecidos. Para crear el juego creado para este proyecto "*Phiby's Toyland Escape*", se ha estudiado la estructura y funcionamiento del juego que le precede y en el cual se basa, "*Phiby's Adventures 3D*".

2.2.2 Phiby's Adventures 3D

La primera versión de "*Phiby's Adventures v1*" se encuentra recogido en el entorno Blexer v1 [16] siendo la captura de movimientos realizada por la cámara Kinect Xbox 360 y creado en Blender. Se basaba en un conjunto de minijuegos, en concreto remar, escalar, cortar y volar o bucear, que mediante el movimiento de los brazos y tronco se controlaba el personaje, pasando de una escena a otra, explorando un mapa y superando los mismos minijuegos. Después se empezó a crear la versión que actualmente está en desarrollo "*Phiby's Adventures 3D*" en Unity y con la Kinect One [17], el cual muestra un mundo abierto dando la posibilidad al jugador de explorar el entorno a través del movimiento de su tronco, pudiendo jugar también mediante teclado. Se mantuvieron los minijuegos iniciales (escalar, cortar, remar y volar/bucear), introduciendo elementos como NPCs (*Non Playable Characters*), para dar mayor dinamismo y causar interés al usuario.

La estructura de este entorno está basada en escenas, cada una siendo un minijuego, que van sucediendo según se avanza en la historia. Se divide en dos capítulos, cada uno con varios *checkpoints* (implementados por Haoru Quin [18]) que implican diferentes objetivos a superar. La escena principal se desarrolla en una isla y los minijuegos de los que consta hasta el momento son los siguientes:

- "***Escape from the cage***": El movimiento a realizar implica mover los brazos de arriba hacia abajo, simulando el acto de escalar. Se debe escalar una jaula en la que está encerrado el personaje hasta que se llegue a un hueco que hay en la parte superior. Es el primer minijuego del juego.
- **"*Climb an apple tree*":** el movimiento a realizar es el de los brazos de arriba abajo, como si se escalara. Se debe escalar un árbol recogiendo manzanas hasta que se tengan las necesarias para rellenar un bol. Es el segundo minijuego del juego.
- **"*Chop Wood for bridge*":** el movimiento a realizar es el del brazo derecho o el izquierdo, según la configuración establecida, de arriba abajo, como si se estuviera cortando. Se deben cortar troncos hasta tener la madera suficiente para construir un puente y cruzar el rio. Es el tercer minijuego del juego.
- **"*Move rocks*":** el movimiento a realizar es el de los dos brazos de un lado a otro, como si se estuvieran apartando objetos. Se deben apartar las rocas caídas de una montaña para liberar a un personaje enterrado en ellas. Es el cuarto minijuego del juego (pendiente de depuración en última versión).

Las escenas anteriormente explicadas pueden visualizarse en la figura 3.

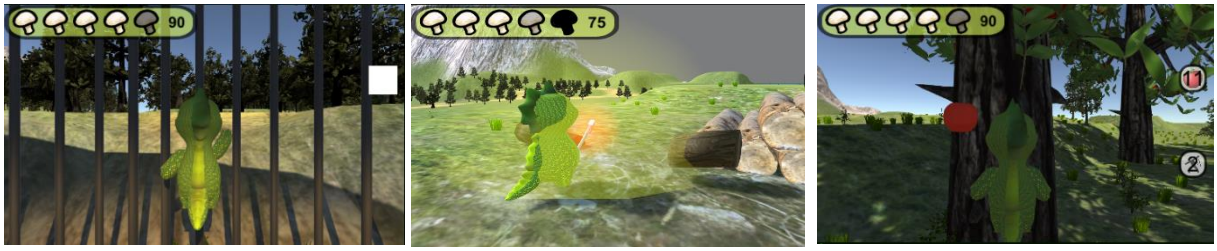

Figura 3. Escenas de los minijuegos implementados en "*Phiby's Adventures 3D*". Izquierda: Escape from the cage. Centro: Climb an applet ree. Derecha: Chop Wood for bridge.

La estructura de este juego fue remodelada por Juan Ignacio Fuentes-Pila [19], quedando el diagrama de flujo de la figura 4.

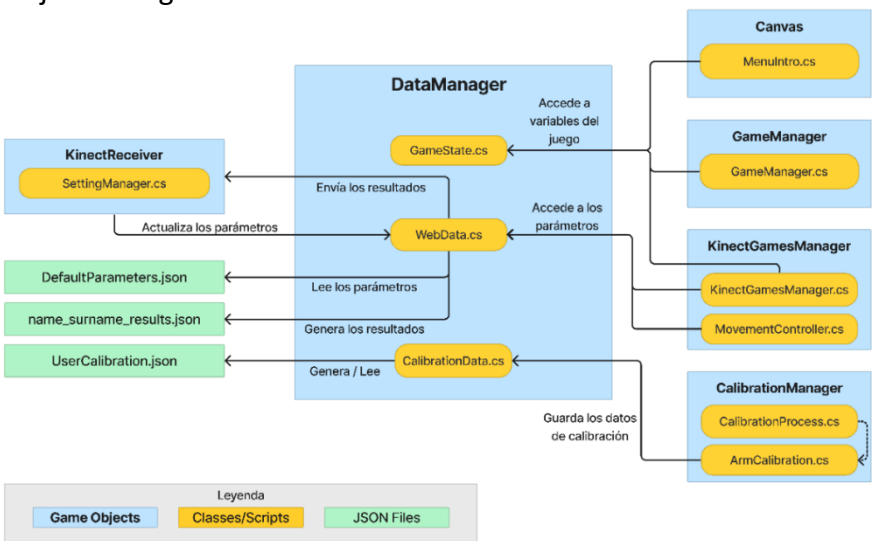

Figura 4. Diagrama de flujo de la estructura de clases y objetos del juego "Phiby's Adventures 3D", extradido del TFG del Juan Ignacio Fuentes-Pila [19]

- El traslado de movimientos y comunicación directa con el middleware se hace a través del objeto ***KinectReceiver***, el cual se encuentra asociado al personaje y que contiene el esqueleto de referencia Kinect. A partir de este y el script *KinectReceiver.cs*, se trasladan los movimientos que se realizan al personaje.
- ***DataManager*** tiene como función manejar la información global del juego en cuanto a la progresión de este.
- ***Canvas***, ***GameManager*** y ***KinectGamesManager*** se encargan de acceder a los datos globales de GameState.cs.
- ***CalibrationManager*** recoge los datos de la calibración y los guarda.

Es este juego en el que se basa el creado en este proyecto, siendo una copia a menor escala con los mismos movimientos a realizar y otros, pero escenarios e historia diferente. La principal diferencia que se va a mostrar es que en el juego base, el jugador puede mover al personaje de forma libre a través de su tronco o el teclado, y en el creado, el movimiento en la escena principal no es libre. Además de esta, el objetivo de este juego es que las escenas creadas para cada minijuego sean modulares e intercambiables en orden y frecuencia de aparición, siendo esta decisión tomada mediante IA (inteligencia artificial). Es por esto por lo que el proyecto sirve como base para la implantación de este mecanismo. A mayores y relacionado con este tema, en un futuro se pretende prescindir de la configuración manual por parte del terapeuta para que sea la IA la que implante estos parámetros. La estructura de *Phiby's Toyland Escape* se asimila a la de este juego explicado, pero ha sido totalmente remodelada.

2.3 Herramientas de trabajo

La principal herramienta que se ha utilizado para el desarrollo de este proyecto es el motor de videojuegos Unity [20], el cual incluye recursos de diseño, modelado, animación, programación y optimización en una sola interfaz. En concreto, para este proyecto se ha utiliza la versión 2021.3.34f1_LTS, siendo versión *LTS* (*Long Term Support*), lo que significa que sufre menos actualizaciones.

Dentro de la plataforma, los desarrolladores pueden crear y controlar mundos virtuales y contenido de forma visual a través de código en lenguaje C++ y C# orientado a objetos. Utiliza un sistema de renderización basado en gráficos en tiempo real para crear imágenes en pantalla, lo que implica renderización de modelos 3D, efectos visuales, iluminación y otros. Además, incluye un sistema de física integrado que simula el movimiento y las interacciones físicas entre objetos en la escena, lo que permite que estas colisiones, se muevan, interactúen con fuerzas externas y respondan a la gravedad de manera realista.

La interfaz de usuario de ésta se divide en ventanas, siendo las principales la de *Hierarchy*, donde se organizan los objetos de la escena abierta, Inspector, donde aparecen los complementos que sostiene cada objeto de la escena, y las ventanas de control como son la de *Scene*, *Console* o *Game*.

Para el control de la lógica se ha utilizado el editor de código Visual Studio 2019 [21], con soporte de depuración para la edición de scripts. En este caso, se ha utilizado el lenguaje C#, derivado de C/C++. Es en esta plataforma donde se crean los scripts que serán usados por Unity para el control de objetos y lógica del juego.

Para la captura de movimientos, como ya se ha mencionado, se utiliza el dispositivo Kinect v2 desarrollado por Microsoft [22]. Este módulo posee una resolución de 1920x1080 con una frecuencia de imagen de 30 fps. y un formato de resolución de 16:9. La distancia mínima de uso es de 1,37 metros y puede realizar la detección simultánea de 6 personas y 25 puntos de cada cuerpo a la vez. Estas características hacen de ella una opción muy interesante para el desarrollo de aplicaciones que impliquen la captura de movimientos de un usuario para su posterior interpretación.

Figura 5. Cámara *Kinect v2* para *Xbox One* por *Microsoft*.

Para el modelado de objetos en la escena, se ha utilizado el software *Blender* en su versión 3.3.3 [23]. Este, contiene herramientas para permitir a los artistas crear modelos 3D detallados y realistas que luego pueden ser importados directamente a los motores de juego como *Unity*. Ofrece, además, un sistema de animación avanzada que permite crear animaciones complejas y realistas para personajes, objetos y entornos. Otro punto importante es su motor de renderizado integrado, que permite renderizar imágenes fotorrealistas y secuencias de animación directamente dentro del software.

3. Especificaciones y restricciones de diseño

Para la ejecución de este proyecto, se tomaron de base dos necesidades principales: un juego no muy extenso y una lógica sencilla de modificar por medio de código, con el objetivo de incorporar inteligencia artificial posteriormente. Con esto, los principales requisitos del trabajo son los siguientes:

- **Configurabilidad del grado de dificultad móvil (Calibración)**

El juego debe ser configurable para diferentes grados de dificultad móvil, permitiendo su calibración según las necesidades del usuario

- **Conexión con la web "*Blexer-med*" para seguimiento de pacientes**

Debe ser capaz de comunicarse con la web para la adaptación de parámetros por parte de terapeutas y la visualización de resultados, conectado a través del middleware K2UM.

- **Conservación de movimientos originales del juego "*Phiby's Adventures 3D*"**

Debe mantener en los minijuegos, los cuatros tipos de movimientos del juego original: escalar, cortar, remar y esquivar.

- **Movimientos predeterminados del personaje principal**

El movimiento del jugador no debe ser libre, sino preestablecido a través de código. Esto es importante porque en los anteriores juegos se ha observado que, al ser un mundo libre totalmente explorable a través de movimientos del tronco, los pacientes presentaban un mayor grado de cansancio y era más difícil de controlar. Por otra parte, debe ser establecido por código para que en un futuro pueda ser modificado más fácilmente.

- **Facilidad de configuración y manipulación por algoritmos inteligentes**

El juego debe estar estructurado y programado de una forma fácil de configurar para facilitar la posterior implementación de la IA.

- **Programación de tiempos de descanso entre ejercicios**

Entre minijuegos debe haber tiempos de descanso modificables para adaptarlos a las necesidades de cada paciente.

- **Diseño de escenarios diversos para cada tipo de movimiento**

Se debe diseñar al menos un escenario para cada tipo de movimiento, con la posibilidad de tener varios escenarios en cada uno y que así en un futuro la IA pudiera decidir cual mostrar y dar mayor dinamismo.

- **Atractivo y entretenimiento para niños**

El juego debe ser atractivo y entretenido, con elementos visuales y mecánicas de juego que los mantenga comprometidos y motivados durante la experiencia.

- **Cumplimiento de función de ejercicio de rehabilitación física**

Debe cumplir con su principal función de ejercicio para rehabilitación física para personas con problemas de movilidad, proporcionando actividades que promuevan el movimiento y la recuperación de manera segura y efectiva.

Estas especificaciones y restricciones de diseño se basan en la necesidad de crear un videojuego terapéutico efectivo y atractivo que cumpla con los objetivos de rehabilitación física y entretenimiento para niños con movilidad limitada.

4. Descripción de la solución propuesta

En base a las especificaciones y objetivos anteriormente mostrados y habiendo hecho un estudio de los proyectos ya existentes en el campo de los videojuegos terapéuticos, se ha desarrollado una solución que cumple con todos ellos. En esta sección se detalla el proceso de desarrollo seguido para la creación del juego "*Phiby's Toyland Escape*", desde la parte más fundamental para entender el juego y su contexto, hasta el funcionamiento interno de los scripts más importantes.

Se proporciona una visión general del concepto del juego y su narrativa, explicando cómo se ha diseñado para ser atractivo y motivador para los niños, a la vez que cumple con sus objetivos terapéuticos. A continuación, se realiza un análisis detallado de cada una de las escenas que componen el juego, describiendo su diseño, su propósito dentro de él y el modo de funcionamiento específico de cada una. Por otro lado, se muestra como está implementada la calibración de este para su mejor accesibilidad para diferentes grados de movilidad.

La conexión entre el juego y la web "*Blexer-med*" es otro componente importante que se describe en esta sección. Se detalla el proceso lógico que permite esta integración, facilitando el seguimiento de los progresos del paciente y la adaptación de los ejercicios por parte de los terapeutas.

Además, se muestra como está estructurada la parte visual (interfaz de usuario) y auditiva de la aplicación, la cual sostiene un papel importante para una buena retroalimentación del usuario y una mejora de la experiencia. Para una comprensión completa de la lógica interna del juego, se hará un análisis de su estructura, acompañando cada línea de trabajo con un diagrama de flujo explicativo.

Por último, se aborda el proceso de modelaje y texturizado de los elementos que componen las escenas.

4.1 Estructura principal

Para desarrollar la idea principal del juego, es esencial centrarse en las siguientes dos restricciones ya mencionadas: el movimiento del jugador no debe ser libre y el juego debe contener los cuatro movimientos del juego original (*Phiby's Adventures 3D*). Con esto, se planteó que debía ser un mundo no muy extenso en el que no fuera posible explorar el entorno y que se desarrollaran minijuegos en contexto con el alrededor con los movimientos establecidos.

Así, se determina que el hilo y contexto inicial del juego es la habitación de un niño, de una edad parecida al público al que está dirigido este proyecto. Este entorno familiar y lúdico pretende despertar el interés y la motivación de los niños por jugarlo.

Con esto, se plantea la siguiente narrativa: un niño está jugando a *Phiby's Adventures 3D* en su portátil cuando, de repente, algo hace que se convierta en el personaje protagonista de este (Phiby). Para poder volver a su forma original, debe recolectar la suficiente energía y

llevar a Phiby de vuelta al portátil. Esta energía es recibida a través de los minijuegos que van sucediendo a lo largo del camino.

Como ya se ha mencionado, el personaje del juego es Phiby, el cual pertenece al juego anterior ya mencionado y el cual se ha extraído íntegramente con animaciones como andar y estado de reposo. Se trata de un modelo con forma de anfibio de carácter amistoso, como puede verse en la figura 6, elegido específicamente para un videojuego donde el público objetivo son niños.

Figura 6. "Phiby", personaje principal de "*Phiby's Adventures 3D*" y de "*Phiby's Toyland Escape*"

Para el control del personaje, se puede realizar de dos formas: por medio de movimientos detectados por la Kinect o a través del teclado.

Controles por Kinect:

Todos los movimientos necesarios para controlar el juego se realizan con la parte superior del cuerpo, con la intención de que se pueda jugar sentado. En concreto, la mayoría de los minijuegos se realizan con el movimiento de los brazos. En este modo, los movimientos que realiza en jugador se copian en el personaje.

Controles por teclado:

El juego también puede ser manejado por medio del teclado. La única tecla que es necesario pulsar es el "*espacio*", ya que controla la mayoría de las funciones, y las flechas derechas izquierda. En este modo, el personaje contiene las animaciones de idle y andar para cuando se traslada de un punto a otro.

Para el cambio entre Kinect y Teclado, solo es necesario darle a la tecla K.

En el apartado 4.4, tras haber explicado exhaustivamente el funcionamiento de cada una de las escenas, se presentará una tabla que resume los controles del juego.

En cuanto a la estructura del juego, existe una escena principal donde se desarrolla la acción principal del juego. Desde esta, al alcanzar ciertos puntos clave, se cambia a escenas individuales que contienen los distintos minijuegos. Cada minijuego tiene su propia escena, la cual se puede acceder también desde el menú principal, en la ventana de selección de minijuegos (se explica más adelante).

4.2 Escena principal

Como ya se ha mencionado, todas las escenas se desarrollan en un entorno ambientado como si fuera la habitación de un niño como puede verse en la figura 9.

Cuando se inicia el juego, lo primero que aparece es el menú principal, que más adelante se explicará en detalle en el punto 4.8, el cual muestra varias opciones, siendo una de ellas "*Play".* La elección de esta opción lleva al personaje a la escena principal para iniciar el juego y le sitúa en la esquina inferior izquierda. Por defecto, al iniciar el juego se comienza en modo Kinect, establecido en la función *start()* de la clase *KinectGamesManager.cs.*

La configuración que se ha planteado como base es la siguiente:

1. Se inicia juego y anda hasta una estantería, salta la escena de *Climb a Shelf*.
2. Al terminar minijuego anda hasta el suelo junto a un puzzle de una casa a través de cajas y baldas (tras mantener la mano levantada), en destino salta escena de *Chop a House*.
3. Al terminar minijuego anda hasta la carretera (tras mantener la mano levantada), salta escena de *Pull a Car*.
4. Al terminar minijuego anda hasta encima del escritorio junto a portátil (tras mantener la mano levantada), salta escena de *PacMan*.

El mapa del entorno de la figura 7 muestra el recorrido que realiza el jugador.

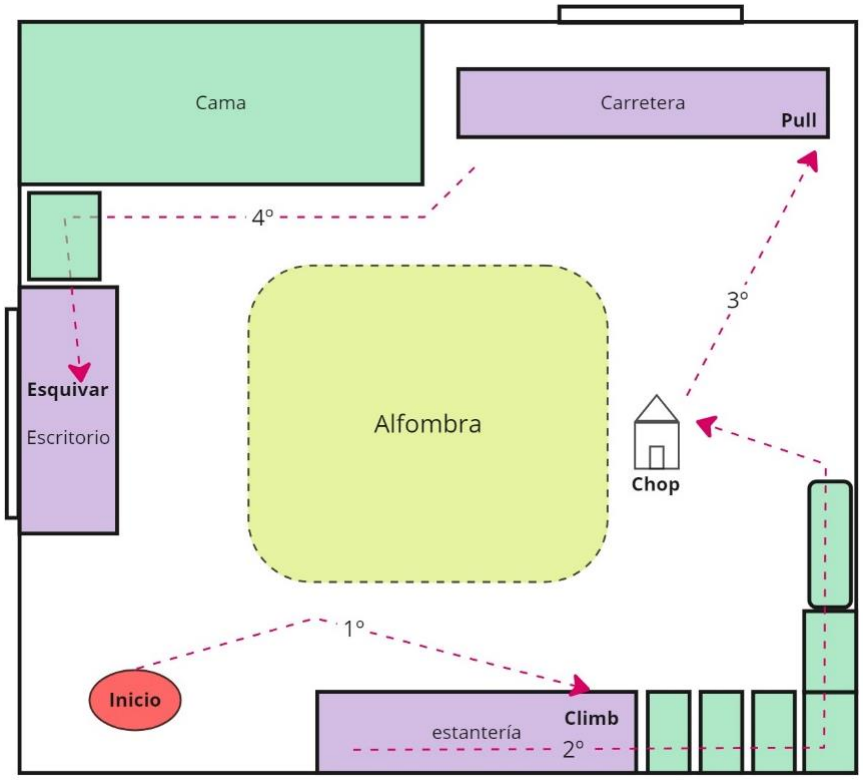

Figura 7. Mapa de la escena principal del juego y el recorrido planteado.

Para iniciar el juego, el jugador debe levantar un brazo. Al realizar esta acción, saldrá una nube azul encima de Phiby, que cambiará de color a verde cuando haya pasado un tiempo determinado, y al bajar el brazo Phiby empezará a andar solo hacia el destino.

Esta funcionalidad se controla desde el script *PhibyWalkStaight.cs* , el cual está asociado a dos *collider*, uno para cada mano, situados de tal forma que cuando el Phiby levante alguno de los brazos entre en este *collider*. Al entrar, comienza un contador de tiempo que se pone a 0 si baja el brazo y deja de estar dentro del *collider*. Cuando pasa un tiempo determinado, en este caso se ha propuesto un tiempo de 3 segundos con el brazo arriba, puede bajarse el brazo y Phiby comenzará a andar. Puede verse en la figura 8 el diagrama de flujo de inicio del juego.

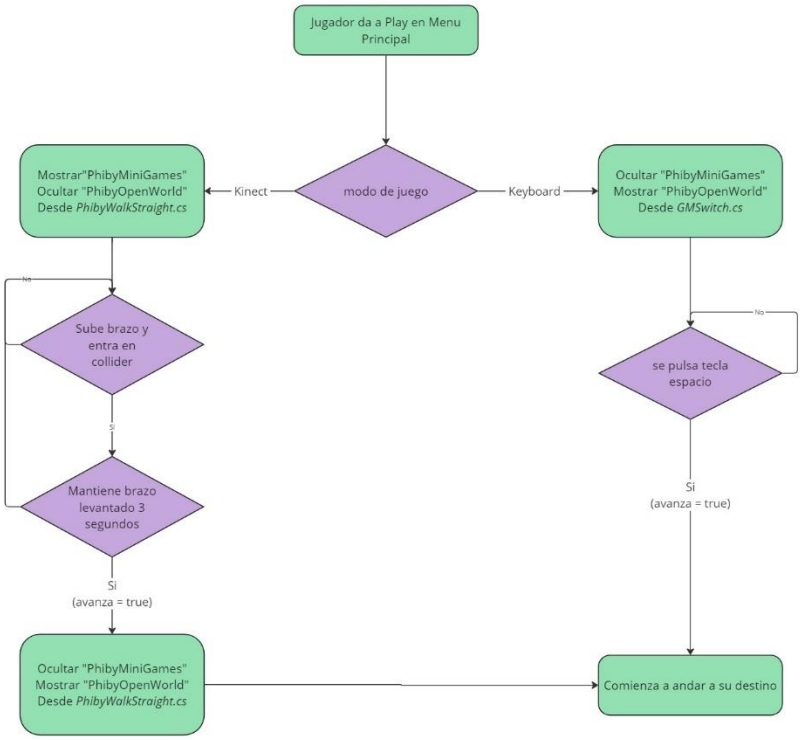

Figura 8. Diagrama de flujo de inicio del juego

Para que esta acción realice su funcionalidad correctamente, es necesario colocar dos modelos del carácter principal Phiby en la escena. Uno se corresponde con el personaje que reproduce los movimientos del jugador y contiene las físicas, es decir, contiene el esqueleto referencia para la Kinect, lo cual se explicará más delante. El otro se asocia al Phiby que no responde a los movimientos de la Kinect y que contiene las animaciones de andar, correr e idle usadas a través de un *player controller*. Esto se realiza porque a la hora de que el personaje comience a andar, si se asocia al Phiby que contiene las referencias de la Kinect, se seguirían moviendo los brazos con respecto a los del usuario y además no mostraría ningún tipo de animación, se trasportaría inmóvil de un punto a otro. De esta forma, cuando empieza a andar, se oculta el Phiby asociado a la Kinect (*prefab PhibyMiniGames*) y se muestra el que contiene las animaciones (*prefab PhibyOpenWorld*). Esta funcionalidad de cambio se controla desde el script ya mencionado *PhibyWalkStaight.cs.*

En caso de estar en modo *Keyboard* para que Phiby comenzase a andar, simplemente habría que dar a la tecla espacio. En este modo, no hace falta cambiar de modelo, ya que este cambio

ya se habría hecho al entrar en el modo *Keyboard* (mediante la teca K) y el Phiby que aparecería en un inicio sería *PhibyOpenWorld*. Este cambio se controla desde el script *GMSwitch.cs,* el cual sostiene la funcionalidad de cambiar de modo de juego del modo que, tras verificar que la tecla ha sido pulsada, cambia el valor de la variable *isKinect* de la clase *KinectGamesManager.cs* al contrario del que estaba. Esta última clase mencionada es la cual alberga todas las variables importantes que no deben ser borradas al cambiar de una escena a otra, por lo que contiene la función *DontDestroyOnLoad()*. Lo que hace esta función es mantener un objeto a través de múltiples escenas, evitando que sea destruido cuando se carga una nueva. De esta clase se hablará en varios puntos para referirse a variables comunes del juego.

Como la acción de andar se realiza de forma automática, el camino que sigue el personaje debe estar preestablecido a través de unos puntos y no es posible cambiar su rumbo mientras se juega (no se puede explorar el entorno). Esta función se realiza a través de *NavMesh.* Es una funcionalidad que posee Unity la cual muestra un mapa de navegación virtual que se genera de manera automática (pudiendo restringir zonas) y que determina por donde puede andar el personaje, quedando delimitada con una red azul como puede verse en la figura 9.

Figura 9. Malla de navegación del entorno calculada (zona azul)

Para habilitar esta función en el proyecto es necesario realizar las siguientes funciones:

- Seleccionar la geometría que se necesita que sea navegable y desde la ventana *Navigation* (*Window > AI > Navigation*), habilitar *Navigation static,* ajustar los parámetros pertinentes y calcular la malla con herramienta *Bake.* Estas ventanas se muestran en la figura 8.

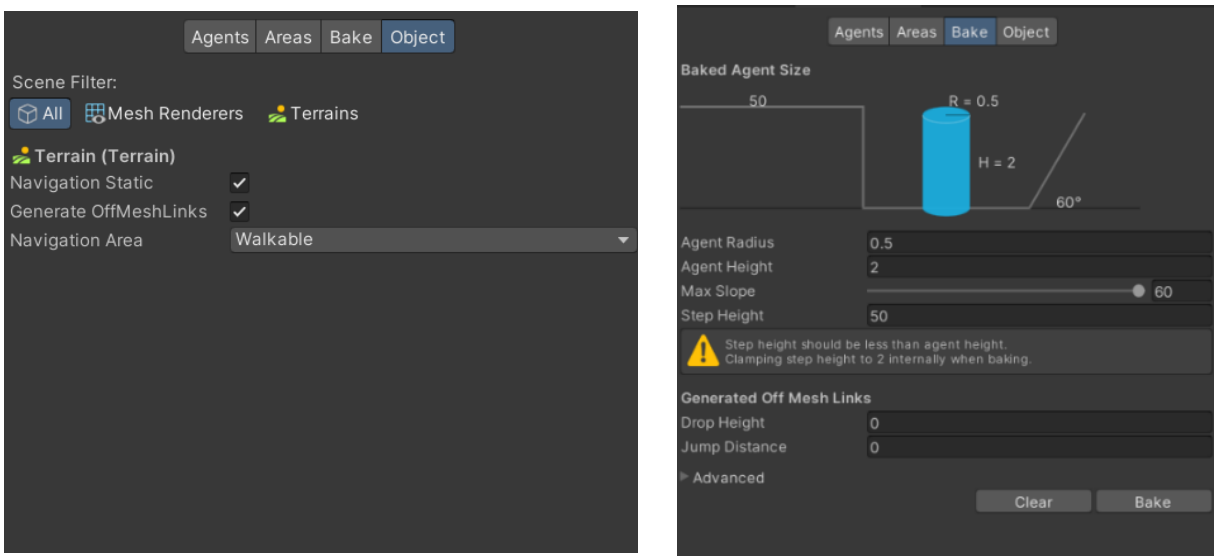

Figura 10.Opciones del objeto(izq.) y parámetros para el cálculo de la malla en la ventana de navegación (dcha.)

- Asignar el componente *NavMesh Agent* al objeto que va a moverse en cuestión, en nuestro caso *PhibyKeyboard* del objeto *PhibyOpenWorld.* El componente se gestiona a través de una serie de parámetros como pueden verse en la figura 11.

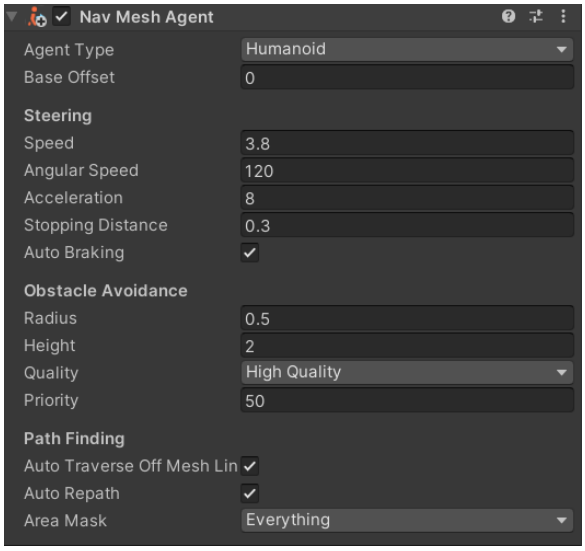

Figura 11. Parámetros del componente *NavMeshAgent.*

Los puntos que son seguidos por el objeto han sido asignados por medio de código desde el script *ControlNavPoints.cs*. En esta, se comprueba en que punto del juego se encuentra el usuario, a través de la variable *checkpoint (KinectGamesManager),* la cual se modifica al salir de cada minijuego con el número correspondiente. En función de esto, se establecen los puntos que va a seguir, modificando el array de puntos llamado *points* que se encuentra asociado a la clase *PhibyMovement.cs,* en concreto son tres puntos los que sigue.

Esta última clase mencionada contiene una variable del tipo *NavMeshAgent* la cual recalcula el camino de un punto a otro y, además, desde esta se lanza la animación de andar.

Todo esto, estando conectado entre sí, comienza a funcionar cuando la variable *avanza* de la clase *PhibyMovement.cs* es puesta a true (tras mantener la mano levantada 3s y bajarla). Esta clase es asociada al mismo objeto que contiene el componente *NavMesh Agent*.

A través de los diagramas de flujo de la figura 12 se pueden comprender mejor estas funcionalidades descritas.

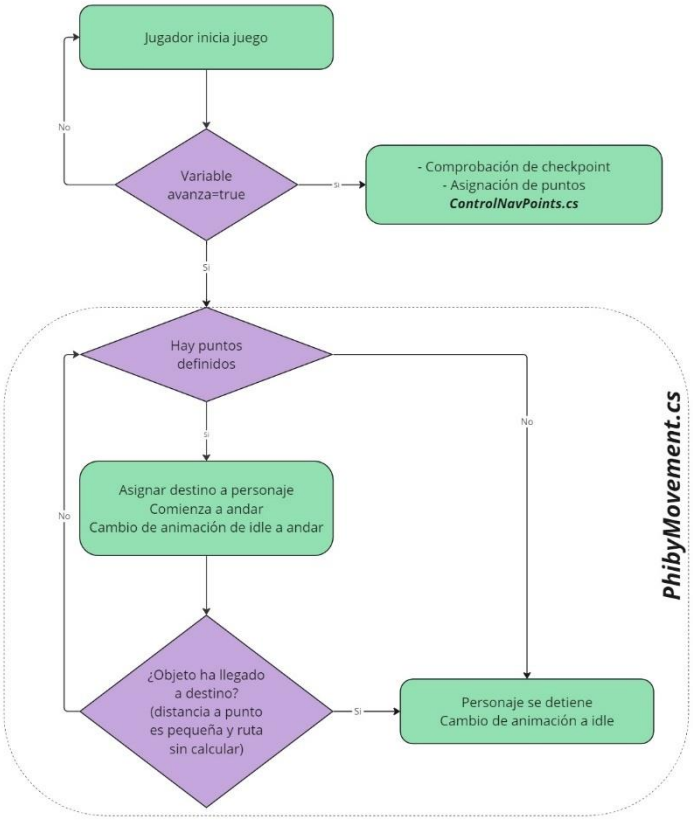

Figura 12. Diagrama de flujo del comienzo de movimiento del personaje.

Al alcanzar el punto de destino, es necesario colocar un objeto vacío (*empty*) que actúe como detector de fin de camino, permitiendo la transición a la escena del minijuego correspondiente. Esto se realiza asociando a este objeto la clase *MiniGameScene.cs.* Cuando el jugador entra en el *collider* del objeto, esta clase verifica el *checkpoint* actual del juego y, según corresponda, lanza la escena del minijuego adecuada.

De los scripts anteriormente mencionados, el relacionado con el movimiento del personaje, *PhibyMovement.cs* ha sido extraído del proyecto "*Phiby's Adventures 3D"* realizando modificaciones para adaptarlo al actual. Los demás mencionados son originales del proyecto.

Lo descrito en este apartado sobre la asignación de puntos y el cambio de escenas es fácilmente modificable a través del código y mediante la adición de objetos vacíos (*empty*) en la escena principal. Para modificar el recorrido del personaje habría que cambiar los valores impuestos en el array de puntos antes mencionado (*points*), según la trayectoria que se quiera

seguir, y al final de esta situar un objeto *empty* para lanzar la escena pertinente. Esta implementación se ha diseñado de esta manera para facilitar la integración de algoritmos inteligentes que puedan alterar el recorrido y el orden de las escenas que se muestran. Esto se corresponde con la restricción de que el juego debe estar estructurado y configurado de forma sencilla y fácil de modificar.

4.3 Minijuegos

Es un requisito que los movimientos a realizar en los minijuegos debían ser los que estaban incluidos en el juego "*Phiby's Adventures 3D*", es decir escalar, cortar, esquivar y tirar.

Se ha planteado para cada uno de estos movimientos una escena independiente, habiendo cuatro escenas, cada una correspondiente a un movimiento y en un lugar del entorno. El paso de la escena principal a los minijuegos se realiza a través de objetos vacíos colocados en la escena, y se han definido cuatro *checkpoints* para controlar el orden de realización de los ejercicios:

- *Checkpoint* 0: Trepar (*Climb a shelf*)
- *Checkpoint* 1: Cortar (*Chop a House*),
- *Checkpoint* 2: Tirar (*Pull a car*)
- *Checkpoint* 3: Esquivar (*Dodge a PacMan*)

En cada minijuego se registran los movimientos realizados para su posterior análisis por parte del terapeuta, además del número máximo de movimientos que deben realizarse. Las variables relacionadas con este recuento se almacenan localmente en los scripts principales de cada minijuego. Por otro lado, las variables que contienen el número máximo de movimientos permitidos y el tiempo máximo disponible son variables comunes, alojadas en el script *KinectGamesManager.cs*, ya que se utilizan con diferentes parámetros según el minijuego que se esté ejecutando.

Si se ha accedido a un minijuego desde el menú principal, se cambia al modo Kinect automáticamente. Cuando termina ese minijuego, además de cambiar de escena, guarda los resultados obtenidos en un documento de texto y los envía a la web si se ha introducido un usuario. Estas funcionalidades se explicarán en el punto 4.7.

Cada minijuego muestra una retroalimentación acústica para una mejora de la experiencia, cuya implementación se explica en el punto 4.10.

4.3.1 Escalar (*Climb a Shelf*)

Objetivo:

Phiby se encuentra en el suelo frente a una estantería de la que cuelga una cuerda. Debe trepar por la cuerda hasta la parte superior de la estantería. Cuando se encuentra más o menos a la mitad del recorrido, llega a una balda en la que no hay objetos, en donde se para y anda de un lado a otro para subir desde el lateral un segundo tramo por otra cuerda que cuelga. Esto se ha hecho para que hubiera un cierto tiempo de descanso entre medias del

juego y que se disminuya la fatiga. El juego termina con el personaje encima de la estantería esperando la orden de andar para trasladarse al siguiente minijuego.

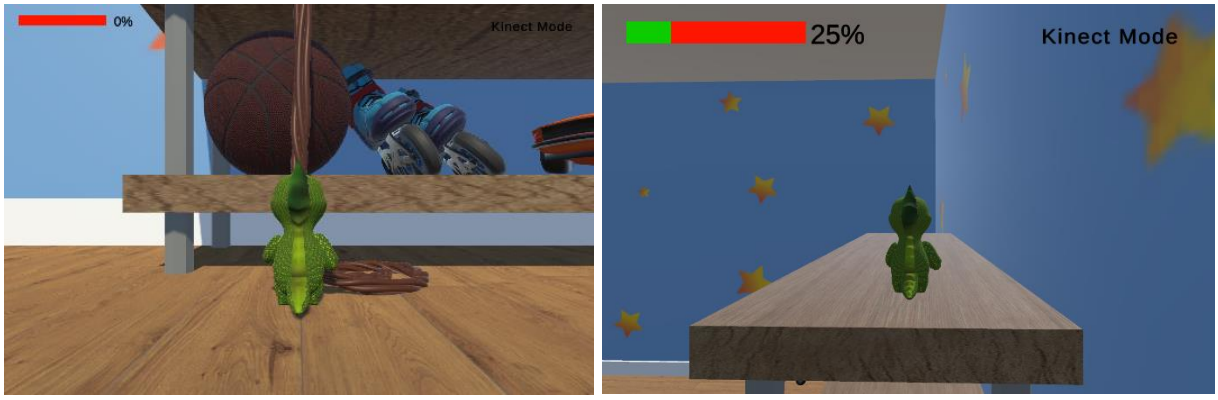

Figura 13. Minijuego *Climb a Shelf*, posición del inicio del juego (izq.); posición del final de juego (dcha.)

Mecanismo:

El usuario debe mover los brazos de arriba abajo, como se visualiza en la figura 14, intercalando derecha e izquierda, como si estuviera escalando una escalera vertical. Esto se ha pensado para que no sea posible avanzar moviendo un solo brazo, sino que se ejerciten los dos brazos por igual. Si está en modo *keyboard*, para avanzar hay que dar a la tecla espacio.

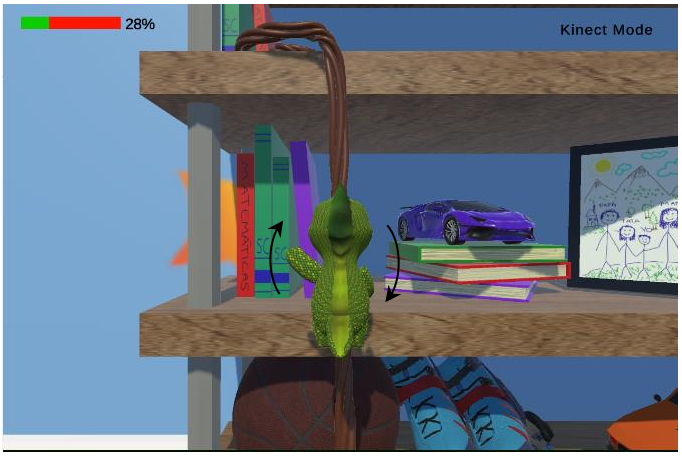

Figura 14. Minijuego *Climb a Shelf* mecánica del minijuego

Lógica:

El script principal que controla el minijuego es *Climb.cs.* Este script está basado en el ya existente *ClimbingTrees.cs* del juego "Phiby's Adventures 3D". Este, está asociado a un *target* en la escena que contiene un *collider*. Cuando la mano que corresponda en el momento entra dentro del *collider*, pone a true la variable *climb* y el personaje se mueve hacia arriba una distancia determinada. Esta distancia se calcula dividiendo el total de la altura que se escala entre el número de movimientos validos que se han propuesto por medio de la web o por los parámetros por defecto.

La parte mencionada, en la que el personaje para a mitad de recorrido y anda solo desde un lado de la balda al otro para continuar subiendo, se controla desde el script

WalkInMiniGames.cs. En este, de la misma forma que se hacía en *ControlNavPoints.cs*, cuando llega a una altura determinada, cambia de tipo de Phiby para ocultar el correspondiente a la Kinect y mostrar el que contiene las animaciones y asigna el punto destino al que debe desplazarse, poniendo la variable *avanza* a true. Además, cambia la posición y rotación de *PhibyMiniGames* al punto de partida del segundo tramo, con el correspondiente *target* de las manos para el funcionamiento de subida. Al llegar el destino, vuelve a cambiar de tipos de Phibys. En la figura 15 se puede ver el inicio, mitad y fin del juego.

Figura 15. Minijuego *Climb a Shelf*, llegada a segundo tramo (izq.); traslado de un lado a otro (medio); posición de inicio de subida del segundo tramo (dcha.)

Parámetros:

Los movimientos que se cuentan son los movimientos totales y los válidos. Los movimientos totales son contados cuando una de las dos manos se levanta por encima del cuello y los movimientos válidos suman cuando el personaje sube (tras entrar en el *collider* la mano correspondiente y poner la variable *climb* a true).

Las condiciones que se imponen para que el juego termine y salte a la escena principal o al menú (según desde donde se haya accedido al minijuego), son las siguientes:

- El número de movimientos totales es igual o mayor al número máximo de movimientos totales que se ha impuesto desde la web o por defecto.
- El número de movimientos válidos es igual o mayor al número máximo de movimientos válidos que se ha impuesto desde la web o por defecto.
- El tiempo dentro del minijuego ha superado el tiempo máximo que se puede estar en esa escena impuesto desde la web o por defecto.
- La posición del personaje ha llegado a la parte superior de la estantería.

Los parámetros que se pueden modificar para variar la dificultad el juego son los mencionados para el cambio de escena: *totalMoves_max, goodMoves_max y time_max.* Como ya se ha mencionado, se encuentran adjuntas al script *KinectGamesManager.cs.*

Al salir de la escena, si se ha accedido al minijuego desde la escena principal, se suma la energía correspondiente a la variable común *totalEnergy* del script *KinectGamesManager.cs* y se cambia el *checkpoint* a 1.

4.3.2 Cortar (*Chop a House*)

Objetivo:

Phiby se encuentra en el suelo, sosteniendo un martillo, junto a lo que parece una casa desmontada y en frente de un clavo. Debe dar golpes a los clavos que se van sucediendo hasta que se acaben. Debe dar un número de golpes determinado por clavo, y cuando este llega a su fin, Phiby se desplaza de un clavo a otro. Cuando el número de clavos llega a su fin, el personaje se aleja de la casa andando y se sitúa en un punto para visualizar una animación de la casa montándose. En la figura 16 se muestra el inicio y el fin del juego.

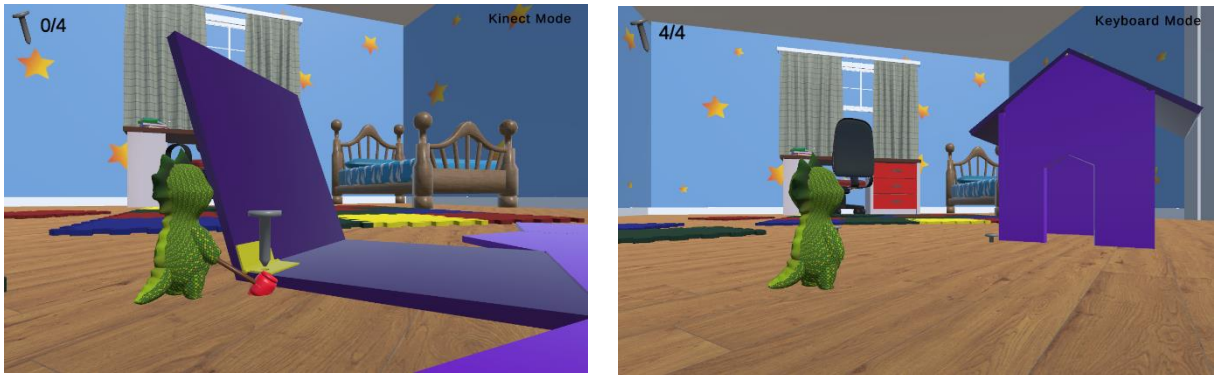

Figura 16. Minijuego Chop a House, posición del inicio del juego(izq.); posición del final de juego(dcha.)

Mecanismo:

El usuario debe mover el brazo derecho de arriba abajo (figura 17), como si estuviera dando golpes a algo. Cuando se levanta el brazo aparece una nube azul que cambia a verde cuando se mantiene el brazo un tiempo levantado. Es con este cambio cuando se puede bajar el brazo y se realiza el golpe al clavo. Si está en modo *keyboard*, para dar el golpe hay que dar a la tecla espacio.

Figura 17. Minijuego *Chop a House* mecánica del minijuego

Lógica:

El script principal que controla el minijuego es *Chop.cs.* Este, está asociado a un *target* en la escena que contiene un *collider*. Cuando el martillo entra dentro de él, aparece una nube de color azul y pone a true la variable *chop* la cual activa un contador de tiempo el cual pasado dos segundos hace cambiar la nube de color a verde. Tras esto cuenta como movimientos

bueno y es posible bajar el brazo para dar el golpe al clavo. Esto solo se produce si se mantiene la mano levantada durante ese tiempo, si se baja la mano y sale del *collider*, la variable *chop* pasa a false y se reinicia el valor del contador a 0.

Tras ponerse la nube a verde y bajar el brazo, el martillo entra en el collider adjunto al objeto clavo, el cual contiene el script *CloveBehaviour.cs.* Este lo que hace es desplazar el clavo hacia abajo una distancia determinada cuando el martillo toca el *collider*. Esta distancia se calcula dividiendo la medida total del clavo entre el número máximo de golpes válidos que se da por clavo, por lo que varía según el valor de ese parámetro que se imponga.

El juego contiene un número fijo de clavos, 4, y a partir de este valor, se distribuye entre estos el número máximo de movimientos válidos que deben hacerse. Los clavos están contenidos en un array de objetos en este script. De esta forma, cuando en un clavo se supera el número de golpes, se cambia de phiby a *PhibyOpenWorld* para desplazarse de un clavo a otro y se oculta *PhibyMiniGames.* Esta funcionalidad, de la misma forma que en el minijuego anterior, se controla desde el script *WalkInMiniGames.cs.* Según el número de clavos que se hayan clavado, se sitúa el Phiby correspondiente a la Kinect en el siguiente clavo junto con el *target* del brazo y se asigna el punto de destino, poniendo la variable *avanza* a true. En la figura 18 puede verse el momento en el que el personaje se mueve de un clavo a otro andando.

Figura 18. Minijuego *Chop a House*, movimiento de un clavo a otro.

Cuando llega al número máximo de clavos, se traslada a un punto alejado de la casa para visualizar una animación creada que consiste en la construcción de esta. Esto se lanza desde el script *HouseAnim.cs* asociado a este objeto.

Esto ocurre cuando el número de golpes válidos ha llegado a su fin y el número total de movimientos y el tiempo no han superado los máximos impuestos. En el caso de que alguna de estas dos últimas variables llegase a su máximo, el personaje se colocaría directamente en la posición de observación de la animación, independientemente del clavo en el que se esté operando, y se lanzaría la animación para posteriormente cambiar de escena.

Parámetros:

Los movimientos que se cuentan son los movimientos totales y los válidos. Un movimiento se cuenta como total cuando el martillo entra dentro del *collider* en el script *Chop.cs*. Un movimiento se cuenta como válido cuando se ha mantenido la mano levantada durante 2 segundos.

Las condiciones que se imponen para que el juego termine y salte a la escena principal o al menú (según desde donde se haya accedido al minijuego), son las siguiente:

- El número de movimientos totales es igual o mayor al número máximo de movimientos totales que se ha impuesto desde la web o por defecto.
- El número de movimientos válidos es igual o mayor al número máximo de movimientos válidos que se ha impuesto desde la web o por defecto.
- El tiempo dentro del minijuego ha superado el tiempo máximo que se puede estar en esa escena impuesto desde la web o por defecto.

La escena cambia cuando se habilita la variable s*ceneChange* desde el script *HouseAnim.cs* que indica que la animación de la casa a terminado.

Los parámetros que se pueden modificar para variar la dificultad el juego son los mencionados para el cambio de escena: *totalMoves_max, goodMoves_max y time_max.*

Al salir de la escena, si se ha accedido al minijuego desde la escena principal, se suma la energía correspondiente a la variable común *totalEnergy* del script *KinectGamesManager.cs* y se cambia el *checkpoint* a 2.

4.3.3 Tirar (*Pull a Car*)

Objetivo:

Phiby se encuentra en el suelo frente a una carretera de juguete y un coche (figura 19). El personaje debe hacer mover unas palancas encima del coche para avanzar y con esto llegar al final de la carretera. Al lado del jugador se sitúa un *NPC* que actúa como oponente en una carrera. Cuanto más rápido se realicen los movimientos, antes se llega al destino y se gana o no al *NPC*.

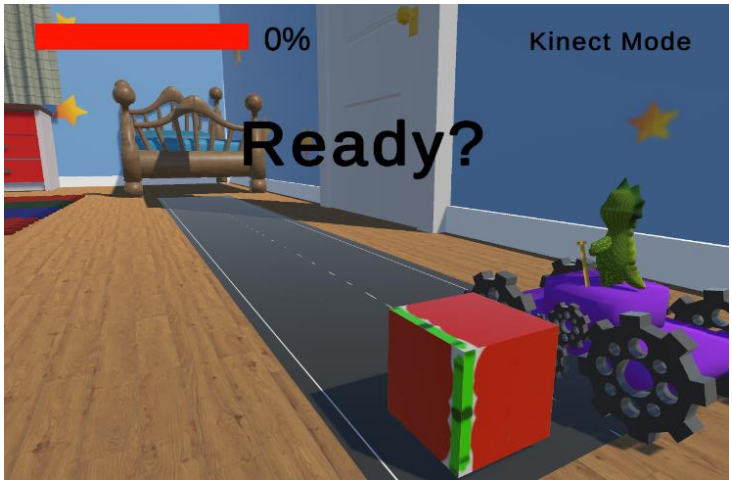

Figura 19. Minijuego Pull a Car, posición al inicio del juego.

Mecanismo:

El usuario debe mover los brazos de delante hacia detrás simultáneamente como si estuviera moviendo una palanca de forma horizontal, como en la figura 20. Si se mueve un solo brazo o los dos de forma descoordinada, el personaje no avanzaría. Esto se ha realizado de esta manera para que se ejerciten los dos brazos de la misma forma. Si está en modo *keyboard*, para avanzar hay que dar a la tecla espacio.

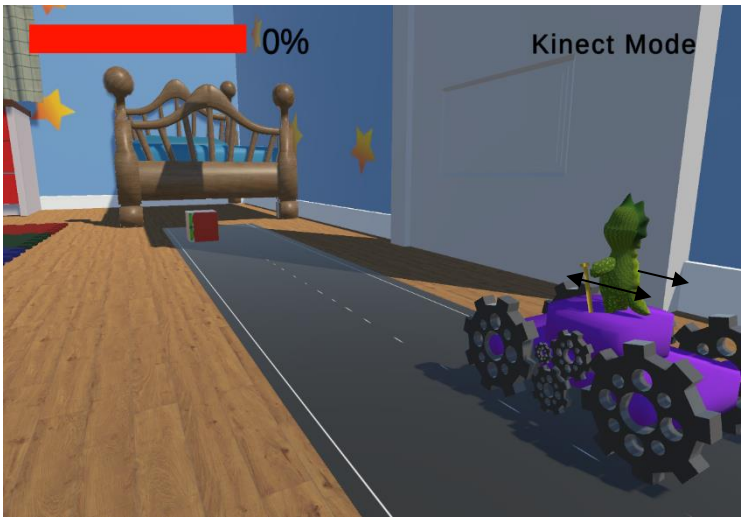

Figura 20. Minijuego Pull a Car mecánica del minijuego.

Lógica:

El script principal que controla el minijuego es *Pull.cs.* El personaje contiene dos *colliders* delante y dos alargados situados en las manos (para que no haya un margen de distancia entre los *colliders* y el personaje, sino podría haber problemas), cada uno de ellos para cada lado. La posición de estos elementos se muestra en la figura 21.

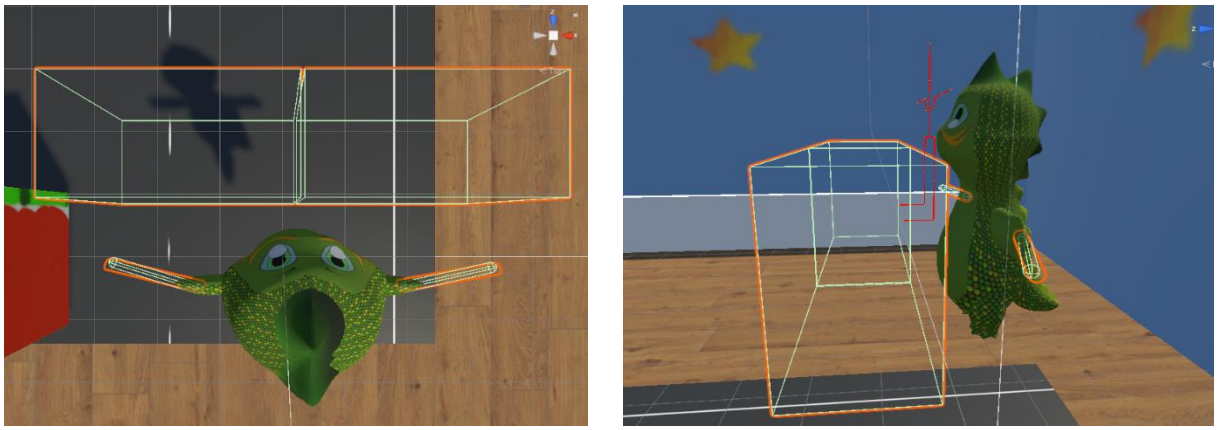

Figura 21. Minijuego Pull a Car, colocación de los colliders para el control de movimiento.

El control de entrada al *collider* de la derecha se realiza a través de *pull.cs* y está adjunto al objeto de situado en la mano derecha. El control de entrada al *collider* de la izquierda se hace desde el script *pullIZQ.cs.* Se ha separado cada lado en scripts distintos porque si se hacía todo desde uno solo, la función *OnTriggerEnter()* ponía a true la mano que antes entrara al *collider* y la otra se quedaba a false, por lo que siempre una iba a ser negativa y nunca contabilizaría el movimiento de las dos manos al mismo tiempo.

Cuando se adelantan las dos manos a la vez (las variables *delante* de cada script son *true*) se pone a true la variable *moving* que hace que el personaje se mueva hacia delante una distancia. Este valor es calculado a partir de la distancia total que se recorre entre los movimientos válidos máximos que deben hacerse.

El control del movimiento del NPC se realiza desde el script *npcRaceMove.cs* el cual, tras mostrar una cuenta de *"Ready, ¡GO!",* comienza a avanzar de forma constante hacia delante hasta el final de la pista.

Parámetros:
Los movimientos que se cuentan son los movimientos válidos que suman cuando las dos manos han entrado a la vez en los *colliders* y el personaje se desplaza hacia delante.

Las condiciones que se imponen para que el juego termine y salte a la escena principal o al menú (según desde donde se haya accedido al minijuego), son las siguiente:

- El número de movimientos válidos es igual o mayor al número máximo de movimientos válidos que se ha impuesto desde la web o por defecto.
- El tiempo dentro del minijuego ha superado el tiempo máximo que se puede estar en esa escena impuesto desde la web o por defecto.
- La posición del personaje ha llegado al final de la pista.

Los parámetros que se pueden modificar para variar la dificultad el juego son los mencionados para el cambio de escena: *goodMoves_max y time_max.*

Al salir de la escena, si se ha accedido al minijuego desde la escena principal, se suma la energía correspondiente a la variable común *totalEnergy* del script *KinectGamesManager.cs* y se cambia el *checkpoint* a 3.

4.3.4 Esquivar (*Dodge Pacman*)

Objetivo:

Phiby se desplaza a encima del escritorio, enfrente del portátil. Al saltar la escena, el personaje que juega no es Phiby, sino el "Come Cocos" del famoso juego *PacMan.* El jugador debe desplazarse de un lado a otro para recoger las cerezas que van cayendo desde arriba. Debe esquivar los fantasmas ya que, si es tocado por uno de ellos, le resta un punto de la puntuación total. El juego termina cuando se han recogido todas las cerezas impuestas. En la figura 22 se muestra la interfaz del juego.

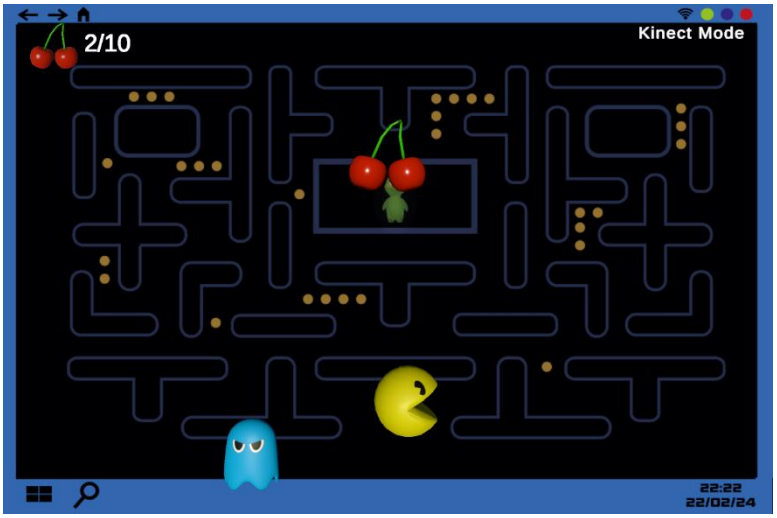

Figura 22. Minijuego *Dodge Pacman*, cerezas y fantasmas cayendo.

Mecanismo:

El usuario debe mover el tronco de derecha a izquierda para dirigir el *Pacman* a donde se quiera (figura 23). Si está en modo *keyboard*, para moverse de un lado a otro se controla desde las teclas "A" y "D" o las flechas derecha/izquierda.

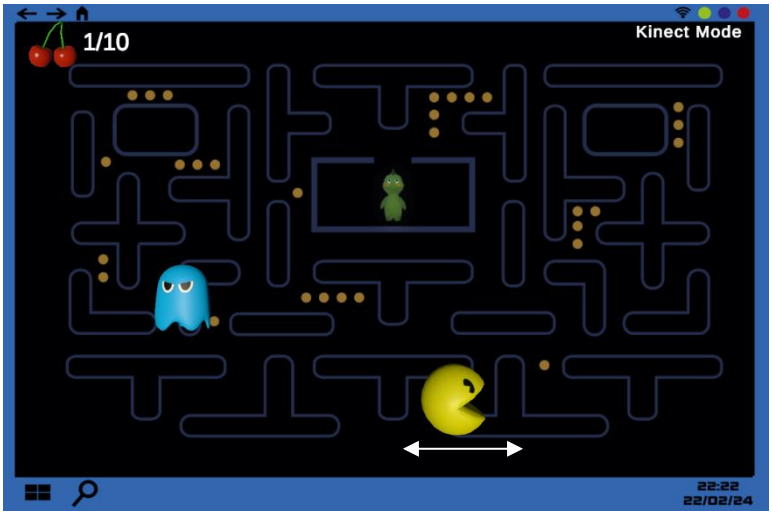

Figura 23. Minijuego *Dodge Pacman*, mecánica del minijuego.

Lógica:

El script principal que controla el minijuego es *Esquivar.cs.* Este script está basado en el ya existente *SpaceShip.cs* del juego "BaseJuegosTerapeuticos" [24].
Este, está adjunto al jugador el cual, a través de la posición del tronco del usuario, dirige el objeto hacia la derecha o hacia la izquierda. En concreto, se controla a través del eje X del *joint* asociado a la cabeza. Además, desde este se controla la recolecta de objetos. Si el jugador toca un objeto con el *tag* "*enemy*", resta un punto a la puntuación si no es menor a 0. Además, cuando el jugador es tocado por un fantasma, cambia su color a rojo por unos segundos, como se muestra en la figura 24. Si no tiene este *tag*, es decir, es una cereza, suma uno a la puntuación.

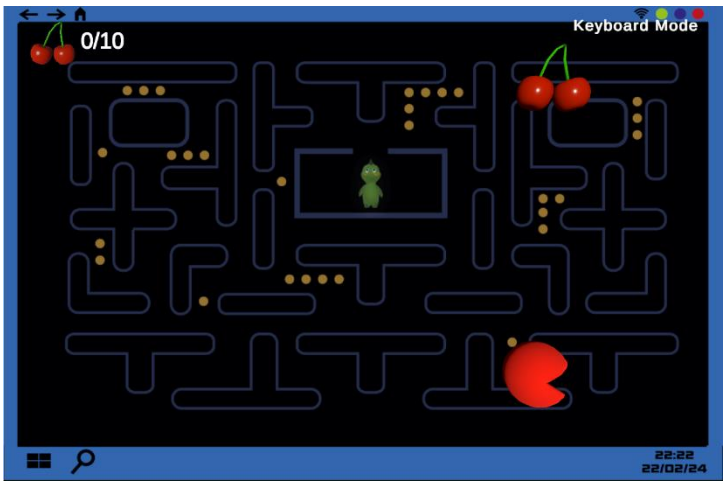

Figura 24. Minijuego *Dodge Pacman*, material del personaje se vuelve rojo cuando es tocado por un fantasma.

El control de la aparición de los objetos desde arriba se realiza desde el script *ObjetivesAvoid.cs*, el cual, desde un array de objetos, lanza un número aleatorio entre 0 y 1 (ya que solo hay dos objetos, fantasma y cereza) y lo presenta desde arriba en una posición aleatoria del eje X con un intervalo de tiempo entre apariciones. A través del script *ObjetivesBehaviour.cs* los objetos se mueven en el eje Y de arriba abajo y al chocar con el jugador o el suelo, estos desaparecen. Los objetos dejan de aparecer cuando se ha llegado al máximo de puntuación impuesto.

Parámetros:

En este minijuego no es posible controlar los movimientos exactos que se realizan, ya que el ejercicio en si es el balanceo del tronco de lado a lado. Por esto, el recuento que se realiza es el de la puntuación total (cerezas recogidas – fantasmas tocados) y el de los fantasmas que se han tocado.

Las condiciones que se imponen para que el juego termine y salte a la escena principal o al menú (según desde donde se haya accedido al minijuego), son las siguiente:

- El número de enemigos que se tocan es igual o mayor al número máximo de enemigos que se ha impuesto desde la web o por defecto (esta variable está contenida en *totalMoves_max* para reciclar variables, ya que esta sino no se usaría).
- La puntuación total (cerezas recogidas – fantasmas tocados) es mayor o igual a la puntuación total máxima que se ha impuesto desde la web o por defecto.
- El tiempo dentro del minijuego ha superado el tiempo máximo que se puede estar en esa escena impuesto desde la web o por defecto.

Los parámetros que se pueden modificar para variar la dificultad el juego son los mencionados para el cambio de escena: *totalMoves_max* (enemigos tocados)*, points_max y time_max.*

Al salir de la escena, si se ha accedido al minijuego desde la escena principal, se suma la energía correspondiente a la variable común *totalEnergy* del script *KinectGamesManager.cs* y se cambia el *checkpoint* a 4.

4.4 Controles

Tras la explicación del funcionamiento de cada minijuego y su control, se presenta a continuación en la tabla 1 un resumen de los controles necesarios en cada situación.

Tabla 1. Controles generales del juego indicando la escena, la acción que provoca y el control en modo kinect y keyboard.

ESCENA	ACCIÓN	KINECT	KEYBOARD
Escena Principal	Iniciar juego/Andar de un minijuego a otro	Levantar mano (mantenerla arriba 3s)	Espacio
Climb a Shelf	Escalar	Mover brazos intercalando derecha e izquierda	Espacio
Chop a house	Dar golpes a clavo	Mover brazo derecho de arriba abajo (acción de cortar)	Espacio
Pull a car	Avanzar hacia delante con un coche	Mover los dos brazos simultáneamente de delante a atrás (acción de tirar de algo horizontalmente)	Espacio
Dodge Pacman	Moverse de izquierda a derecha	Mover tronco de derecha a izquierda	A/←: izquierda D/→: derecha
-	Cambio de modo de juego(*Kinect*/*keyboard*)	K	

4.5 Calibración

Para permitir la adaptación de los movimientos a las capacidades individuales de los usuarios, se ha añadido la opción de calibrar el juego según las necesidades de cada persona. Esta funcionalidad se ha extraído del juego creado por Eduardo Botija "Base para juegos Terapéuticos" [24], por lo que queda pendiente de revisar como trabajo futuro, ya que es una base.

Dentro de la escena de calibración, van sucediendo una serie de instrucciones que el usuario debe realizar con sus brazos y tronco.

La primera requiere levantar la mano por encima de la cabeza para comenzar con la calibración. Este mensaje aparece en la pantalla y cuando se cumple, se oculta y aparece la primera instrucción.

El primer paso de la calibración consiste en inclinar el tronco delante, atrás, derecha e izquierda. Cada vez que completa un lado del tronco, aparece en la pantalla según corresponda derecha/izquierda/delante/detrás. Cuando se realice, se pasa a la siguiente instrucción, en la cual se debe de abrir los brazos en horizontal con la máxima amplitud y moverlos de arriba abajo sutilmente. Cuando se completa esta orden, se cambia de nuevo al menú principal.

Para el control de estas funcionalidades, hay dos scripts principales: *calibrationController.cs* y *calibrationManager.cs.*

- *calibrationController.cs:* script encargado de realizar la toma de parámetros de posición de las articulaciones involucradas en cada uno de los pasos.

  Para la calibración del tronco del jugador, se usan las siguientes funciones:

  - *startSpineCalibration():* función que inicializa los parámetros necesarios para la calibración de los movimientos del esqueleto, *OrgXSpine* (movimiento lateral) y *OrgZSpine* (movimiento delante/atrás). Además, inicializa la variable de inicio de calibración.
  - *calibrateSpine():* función encargada de guardar durante el tiempo de calibración los valores máximos alcanzados en los ejes X y Z.
  - *endSpineCalibration():* función que toma los valores recopilados para trasladar el movimiento del jugador en el entorno real al sistema de coordenadas del juego.

- *calibrationManager.cs:* script encargado de controlar lo relacionado con las indicaciones mostradas por pantalla y los tiempos asignados a cada instrucción de la calibración. Además, desde este se llaman a las funciones que correspondan en cada momento del script *calibrationController.cs.*

Para la aplicación de esta calibración, se usan las clases *KinectReceiverAmpl.cs, IKManager* y *Amplifier.cs.* Estas clases quedan pendientes de revisión, ya que se han extraído exactas de las creadas por Cesar Luances del juego "*Phiby's Advetures 3D*".

4.6 Conexión con el Middleware

Como se ha mencionado con anterioridad, en este proyecto se va a utilizar el nuevo middleware creado por Filip Racki que incluye la posibilidad de cambiar entre cámara Kinect para *Xobox One* o *Azure*. Para esto, era necesario la creación de una nueva estructura de captura de información por parte de Unity, que Filip creó como base para los posteriores proyectos.

Para hacerlo más legible, se cambió la implementación de transferencia de un formato propio que sostenía el "*Phiby's Adventures 3D*" a JSON.

La arquitectura actual funciona de la siguiente manera: los datos se obtienen del middleware utilizando la clase *UDPReceive.cs,* la cual emplea UDP para la comunicación. Estos datos se envían al script *KinectReceiver.cs* donde se convierten de JSON a un diccionario de *MJoint*, un objeto que almacena información sobre la posición y rotación de una articulación.

Después, los datos obtenidos se asignan a los objetos configurados en el editor de Unity mediante la clase *KinectReceiverEditor.cs,* que gestiona la interfaz de usuario del editor en Unity. Con la ayuda de la clase *DrawBone.cs* se puede visualizar el esqueleto en escena. La información sobre los objetos físicos en Unity se almacena en *JointObject*. Además, la clase

ApplyJointRotation.cs permite asignar la rotación de una articulación a cualquier tipo de objeto en Unity, lo que se utilizarla para asignar las rotaciones de las articulaciones del personaje en el juego. A la hora del envío de resultados, el middleware almacena los resultados de los ejercicios en la memoria interna del dispositivo y, al finalizar el juego, estos datos se envían al servidor donde se almacenan. Para este envío, se utiliza la clase *UDPSend.cs,* la cual sostiene el método *sendString()* el cual, a través de su llamada, envía los datos al *middleware*.

En la figura 25 se resume la estructura de la conexión por parte de Unity con el middleware.

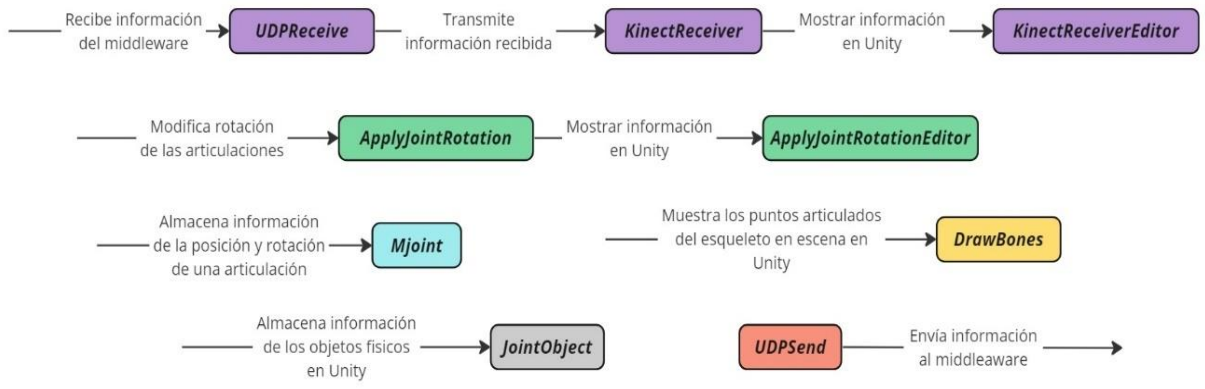

Figura 25. Principales clases que componen la captura de movimientos y traspaso de información de Unity al Middleware.

En cuanto a la estructura interna del middleware, Filip reestructuró el código en clases más pequeñas cada una con una responsabilidad y con una extensión limitada para mejorar la legibilidad y facilitar las futuras modificaciones. Además, diseñó una nueva interfaz de usuario en la cual se muestran las ventanas:

- ***LoginView***: para el inicio de sesión y acceso a la página del administrador
- ***MainView***: para la selección del dispositivo Kinect y monitorear su estado.
- ***CameraView***: para mostrar la imagen de la cámara y la superposición del esqueleto.
- ***LogView***: para mostrar los datos en formato JSON que se transmiten.
- ***SettingsView***: para la configuración de la aplicación.

Para más información de la estructura interna del middleware, puede consultarse la memoria de prácticas de Filip Racki [15].

4.7 Conexión con web Blexer-med, gestión de la configuración

Para la adaptación del juego a las capacidades de cada persona por parte de un terapeuta, es necesaria la conexión con la web *Blexer-med*, la cual se realiza por parte del middleware. Cuando se introduce un usuario, que está registrado en la web, en K2UM, esto es comunicado al juego y asocia los parámetros que correspondan.

Este control se realiza a través del script *SettingsManager.cs.* En este, lo primero que se comprueba es si existe un archivo que contenga los parámetros por defecto (los utilizados si

no se ha introducido ningún usuario). Si no es así, crea un archivo con los valores de cada minijuego en la carpeta *Assets* del proyecto de Unity (instrucción *Application.dataPath*) en un archivo de tipo JSON con el nombre *DefaultParameters.* Esto, se guarda también a través de la función *saveExercise()* en la clase *WebData.cs* para que, en caso de que no se introduzca ningún usuario, se utilicen estos parámetros por defecto. Esta clase, guardará esta configuración en un formato legible durante el tiempo de ejecución de juego.

Tras esto, se espera a que se reciba la información que se guarda en la variable *configuración* de la clase *KinectReceiver.cs.* Si esta variable está vacía, se llama a la función *enviarSolicitudConfiguracion()* para solicitar al middleware la configuración de la web a través de un mensaje de *SettingRequest* cuyo parámetro más importante es "SR", además de las cabeceras del juego y los tipos. En cuanto se produce la conexión entre el middleware y Unity, en *KinectReceiver.cs* se recibe esta información, la cual, si comienza con la cabecera de los juegos con Kinect (*#K2UMUDP*) significa que se ha introducido un usuario que contiene una configuración para el juego. Si no es el caso y *configuración* es vacío, significa que no se ha producido un *login* y se cambia esta variable de vacío a "*null*". Volviendo al script *SettingsManager.cs,* si *configuración* no cambia de nulo a otro valor en un número de veces que se ha intentado requerir esa información, se deduce que no está el middleware funcionando y que debe arrancar de nuevo el software. Si cambia su valor y es "*null*", como ya se ha dicho antes, no hay usuario y se indica a través de la variable *userEntered = false.* En este caso, se utilizarían los parámetros por defecto anteriormente guardados en *WebData.cs*. Si es diferente a "*null*", sí hay un *login* y llama a la función *procesarConfiguracion*() para cambiar los datos de *WebData.cs* por los actualmente recibidos.

A través del diagrama de flujo de la figura 26 se puede observar con más claridad esta funcionalidad de conexión.

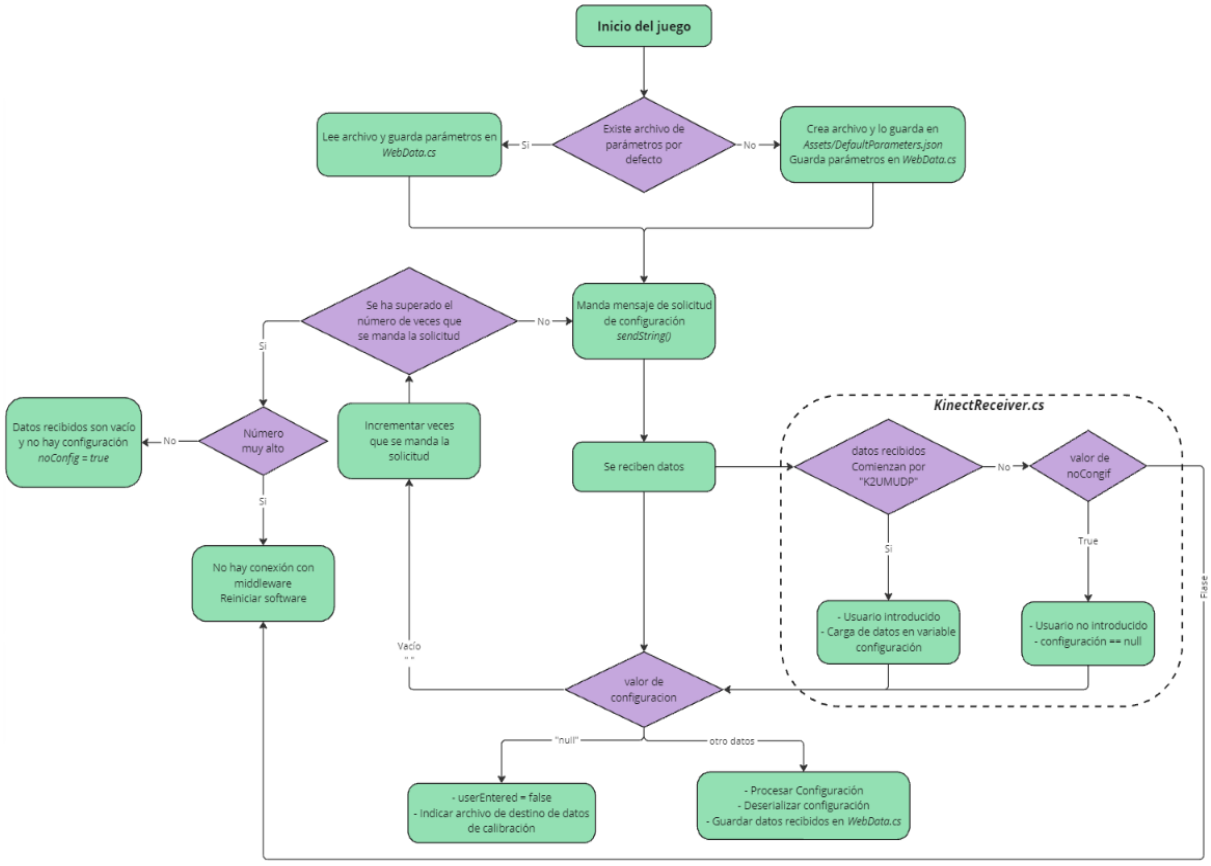

Figura 26. Diagrama de flujo de la captura de configuración.

Si en el middleware no se ha iniciado sesión, al inicio del juego aparecerá un mensaje en el cual da la posibilidad de introducir un nombre con el cual se guardarán los resultados generados. Esta funcionalidad se gestiona a través de los scripts de control del menú principal posteriormente explicado.

En los siguientes apartados se analizará la asociación de estos parámetros recibidos a través y el posterior envío de resultados a la web.

4.7.1 Asociación de parámetros

Si se quieren modificar los parámetros que utiliza cada minijuego, lo primero que es necesario realizar es configurar un nuevo juego en la web *Blexer-med*. Se debe entrar como Super Administrador y configurar un nuevo juego indicando el código a través del cual se va a acceder a él, el nombre y una pequeña descripción.

Con esto, ya se puede entrar en la creación de cada uno de los minijuegos. En este caso se han configurado cuatro, en cada cual hay que indicar el alias por el cual se va a acceder a él, el nombre del juego, una descripción y los parámetros que se quiere poder modificar. En la figura 27 se pueden ver los minijuegos añadidos y los parámetros que cada uno sostiene.

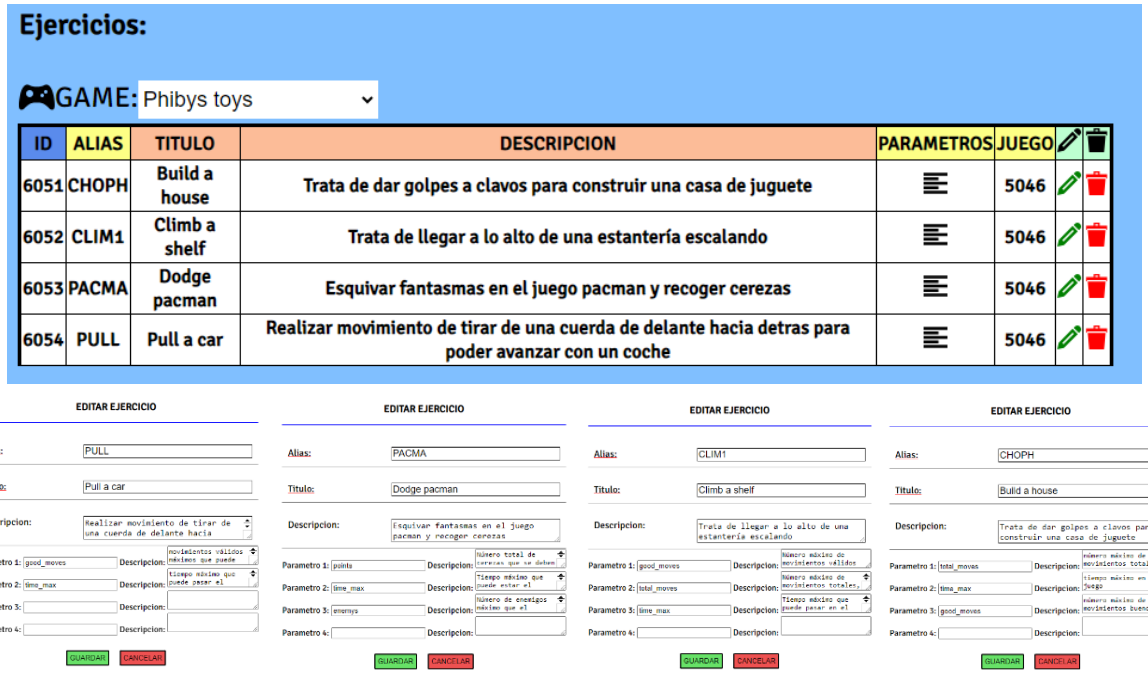

Figura 27. Página web *Blexer-med*. Minijuegos creados para *Phiby's Toyland Escape* (arriba); Parámetros modificables de cada minijuego(abajo).

Tras esto se puede pasar a configurar como terapeuta los valores de cada uno de los parámetros que se han asociado a los minijuegos. En la figura 28 puede verse un ejemplo de la asociación de los parámetros en la web a un paciente anteriormente creado.

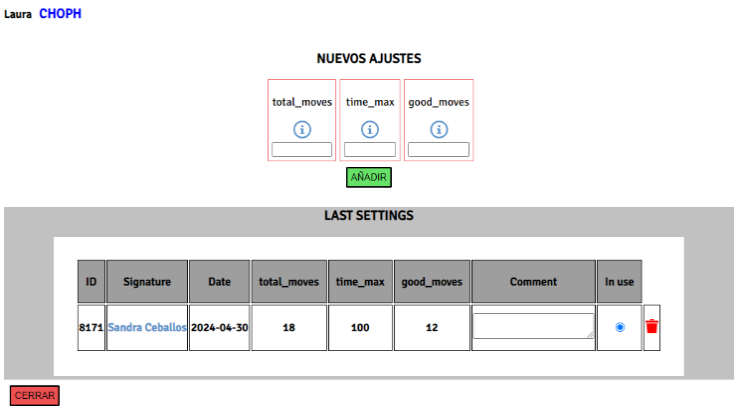

Figura 28. Página web *Blexer-med*, asociación de parámetros para el minijuego *Chop a House*

El siguiente sería lo explicado en el paso anterior, la conexión del middleware con la web y carga de esa configuración en el proyecto de Unity. Estos parámetros, como ya se ha explicado, se guardan en la clase *WebData.cs* en una serie de variables que se mantienen durante el juego. Para poder utilizar estos valores en cada minijuego, es necesario traspasarlos a cada script principal de las escenas individuales. Esto se realiza en la clase

KinectGamesManager.cs. Al inicio de cada minijuego, cambia el valor de la variable *id_game* al que corresponda y se llama a la función *assignParameters(string id_game)* con el ID propuesto. Cada uno de los minijuegos tiene un identificador siendo los siguientes:

- ***Climb a shelf*:** CLIMB1
- ***Build a House:*** CHOP1
- ***Dodge pacman:*** PACMAN
- ***Pull a car:*** PULL

Al llamar a esta función, según el ID, se asignan los parámetros guardados en *WebData.cs* a las variables globales del juego: *totalMoves_max, goodMoves_max, time_max y points_max.* Así, en los scripts principales de cada escena, se cogerán estas variables globales guardadas en *KinectGamesManager.cs* como ya se ha explicado.

A la hora de implementar algoritmos inteligentes al juego, se podría realizar desde la clase *KinectReceiver().* Tras comprobar si hay un usuario introducido, cuando se iguala la variable *configuracion* a la información recibida (variable *data),* debería contener los parámetros creados por la IA. Esa información debería mantener la misma estructura que sostiene la que se recibe por parte del middleware.

4.7.2 Envío de resultados

Al acabar cada minijuego, se llama a la función *saveExerciseResults()* de *WebData.cs* para guardar los resultados obtenidos en la web o, en caso contrario de no haber introducido usuario, en el archivo local del dispositivo.

A esta se le pasan los parámetros obtenidos en el trascurso del minijuego y con ellos crea una variable del tipo *ExerciseResult* que engloba toda la información necesaria referente al juego. Tras la asociación de cada uno de los parámetros a esta variable, se guarda en un archivo en la carpeta *Assets* con el nombre del usuario introducido más "*_results.json*" en formato JSON. Si no hay usuario introducido y no se ha introducido ningún nombre al inicio del juego, el nombre se reduciría a "*_results*". En tal caso de que se halla introducido un usuario, se llama a la función *sendToWeb()* de *KinectGamesManager.cs* con estos resultados y en ella se manda a la web a través de *sendString()* en formato JSON y junto con las cabeceras del juego y el parámetro "FR".

4.8 Interfaz de usuario

A la hora de diseñar un juego, la retroalimentación del usuario es lo más importante a la hora de su ejecución, ya que es el elemento que fomenta la motivación y despierta el interés por seguir jugando. Es por esto, que la UI (interfaz gráfica de usuario) es un componente crucial ya que actúa como puente entre el usuario y el sistema.

4.8.1 Menú principal

El elemento principal de este es el "Menú principal", en el cual se muestran las opciones que puedes realizar. Al iniciar del juego, aparece el nombre de éste y seguido, si no se ha iniciado sesión en el middleware, aparece un formulario indicando que está entrando en el juego sin un usuario introducido, teniendo la posibilidad de meter un nombre para el archivo que guarda los resultados (figura 29).

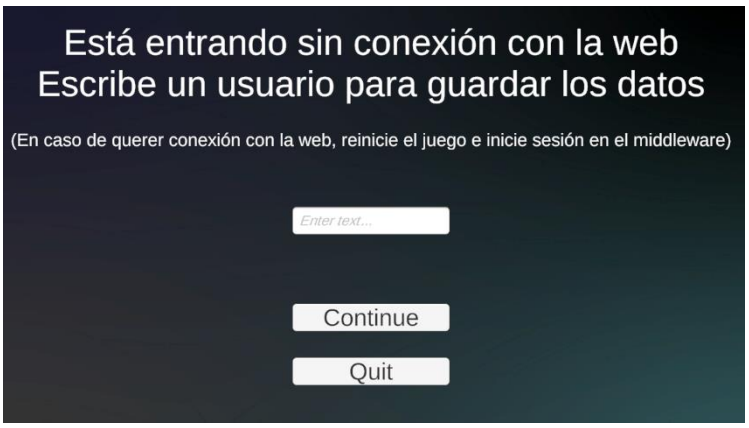

Figura 29. Formulario principal al iniciar el juego.

Es posible darle a continuar sin introducir este nombre, ya que simplemente a la hora de guardar los resultados, el archivo se guardará con el nombre *"_results.json"* (explicado en el punto 4.7.2).

Si se ha introducido un usuario en el middleware, esta pantalla intermedia no aparecerá y se pasará directamente al menú principal correspondiente a la escena en Unity "*MainMenu*". Es importante recalcar que es en esta escena en donde se crea el objeto *KinectGamesManager* al que están asociados los scripts *KinectGamesManager.cs, WebData.cs* y *SaveData.cs.* Esto es porque, al ser la primera escena que se carga y al tener estos la función *DontDestroyOnLoad(),* este objeto no se destruye para las siguientes escenas no es necesario crear en cada escena un elemento con estos scripts. Con esto, en el menú principal se muestran varias opciones como puede verse en la figura 30.

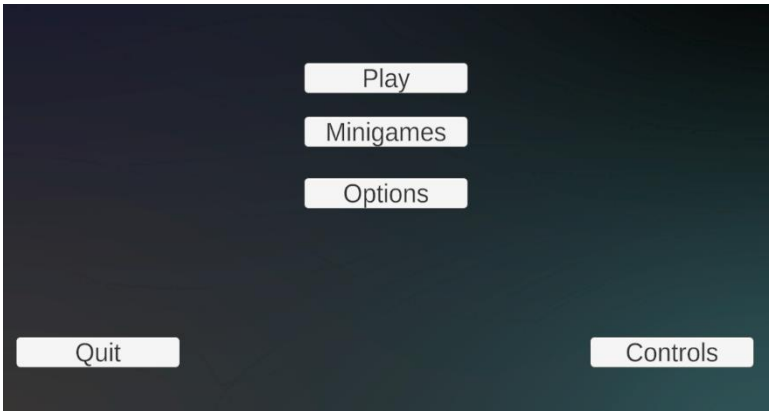

Figura 30. Ventana de menú principal.

Las opciones que se muestran son las siguientes:

- ***Play***: Eligiendo esta opción comienza el juego con su recorrido principal propuesto.
- ***Minigames***: a través de esta se puede acceder a los minijuegos disponibles directamente sin pasar por el resto del recorrido en la escena principal. Al terminar cada minijuego se vuelve a esta ventana de selección. Esta ventana se muestra en la figura 31.

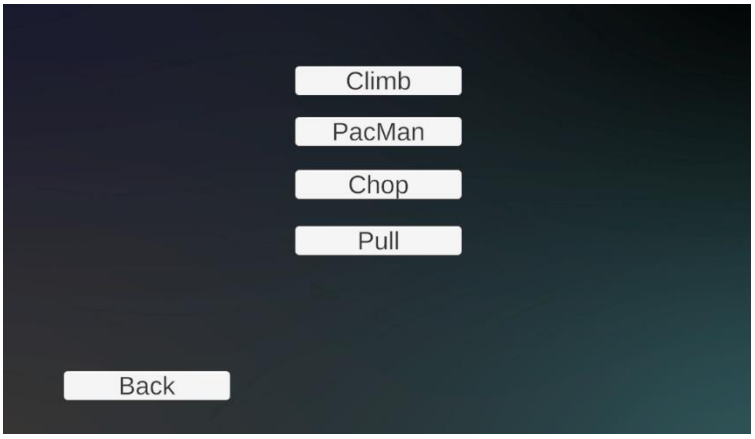

Figura 31. Ventana de selección de minijuegos.

- ***Options***: en esta se muestran las opciones de: calibrar el juego, ver la historia principal del juego y los créditos. Esta ventana se muestra en la figura 32.

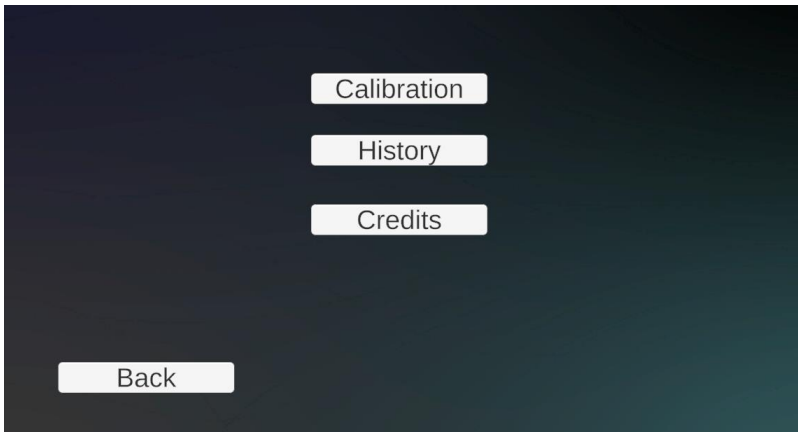

Figura 32. Ventana de opciones.

- ***Controls***: en esta se muestran los principales controles que se necesitan para manejar el juego, incluyendo los correspondientes con la Kinect y los del teclado (*keyboard*). Esta ventana se muestra en la figura 33.

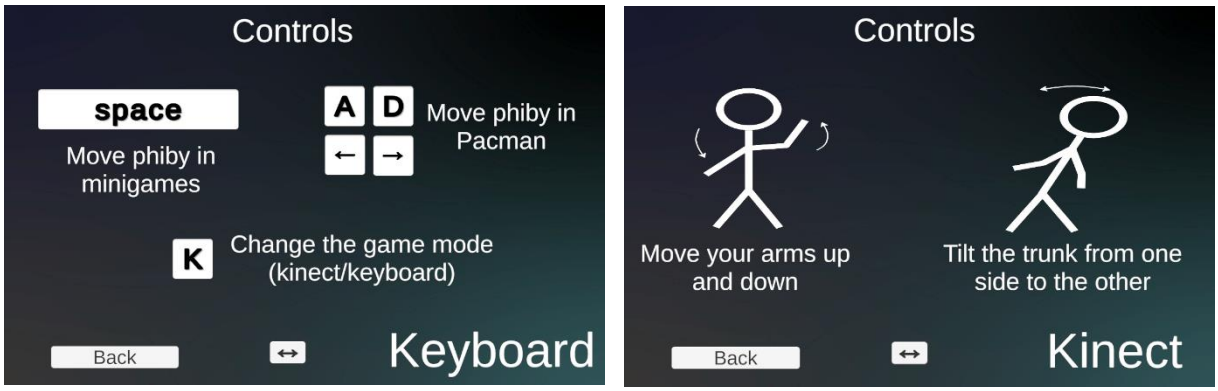

Figura 33. Ventana de controles para el modo Keyboard (izq.) y para el modo Kinect (dcha.).

- ***Quit***: opción que da la posibilidad de cerrar el juego.

Todas estas funcionalidades se controlan desde el script *MainMenu.cs* asociado al objeto en la escena de Unity *MenuManager.* En este, se crea para cada panel un objeto que después debe asociarse a su correspondiente elemento *canvas*. Al inicio del juego, comienza una co-rutina que, pasados tres segundos, oculta el título del juego y muestra el formulario ya mostrado. A la hora de mostrar cada uno de los paneles se utiliza la función *SetActive()* adjunta a los elementos del tipo *GameObject* con las opciones de false (panel oculto) y true (panel visible).

4.8.2 Barra de energía

Para que el usuario reciba retroalimentación por parte del progreso del juego, se ha añadido una barra de energía en la escena principal y en cada minijuego de forma independiente.

Esto se controla desde el script *EnergyController.cs* en el cual, dependiendo de la escena en la que se encuentre, muestra un valor u otro. Para la barra corriente de energía, la cual va cambiando de color de rojo a verde según el progreso, es necesario añadir un elemento slider en la escena con un componente de texto al lado para mostrar el porcentaje de progreso.

A la salida de cada minijuego, esta barra avanza un 25% desde el 0 hasta el 100%. En el caso de los minijuegos de *Climb a Shelf* y *Pull,* el valor que se avanza por cada movimiento válido es calculado al inicio de su script principal como el total (cien por cien) entre el número de movimientos válidos impuestos a realizar. Cuando un movimiento válido es realizado, se suma a esta barra el valor calculado hasta llegar a su fin.

En el caso del minijuego *Chop a House* no existe barra de progreso, sino el número de clavos que se han clavado hasta el momento frente al máximo que se debe conseguir. En este caso, no es necesaria una barra slider en la escena, sino un elemento de texto que desde código cambia su valor. Parecido ocurre para el minijuego *PacMan,* el cual muestra el número de puntos totales que se han conseguido hasta el momento frente a os puntos máximos que se deben conseguir.

Por último, el elemento que todas las escenas, menos el menú principal, comparten es la indicación si se está jugando en modo Kinect o en modo *keyboard*. El cambio de estos letreros se cambia desde el script *GMSwitch.cs.*

4.9 Sonidos

Para ayudar a la realimentación acústica del juego y mejorar la experiencia, se han añadido una serie de sonidos a cada escena que compone el juego.

- **Menú principal**: música de ambiente [25] y sonido de *click* al dar a los botones [26]
- **Escena principal**: (por concretar)
- **Minijuegos**: música de ambiente diferente a la del menú [27]
  - *Climb a shelf*: cada vez que se hace un movimiento bueno, se reproduce un sonido de *check* [28]. Cuando se realiza uno malo se reproduce un sonido que da a entender el error [29]
  - *Chop a house*: cada vez que se da un golpe al clavo se reproduce un sonido de golpe [30]
  - *Pull*: cuando se produce un movimiento bueno y el personaje avanza, se reproduce un sonido parecido al de un juguete electrónico moviéndose [31]
  - *PacMan*: la música de fondo es diferente a los otros, es una melodía retro realizada con ordenador [32]. Cada vez que una cereza es recogida, se reproduce un sonido como el de recoger monedas en el famoso juego Mario Bros [33], y cuando un fantasma es tocado se reproduce un sonido que da a entender el error [34].

Para introducir estos sonidos en la escena de Unity, es necesaria un objeto en el que esté asociado el componente *Audio Source.* Para las melodías de fondo, estos componentes se han añadido en el objeto llamado *Entorno* en el caso de los minijjuegos, y se le ha añadido en el apartado *AudioClip* el audio correspondiente, ajustando además el volumen y la opción de que se reproduzca en bucle. En el caso del menú principal, este componente se encuentra en el objeto *MenuManager.* En la figura 34 puede verse el componente en el inspector de Unity.

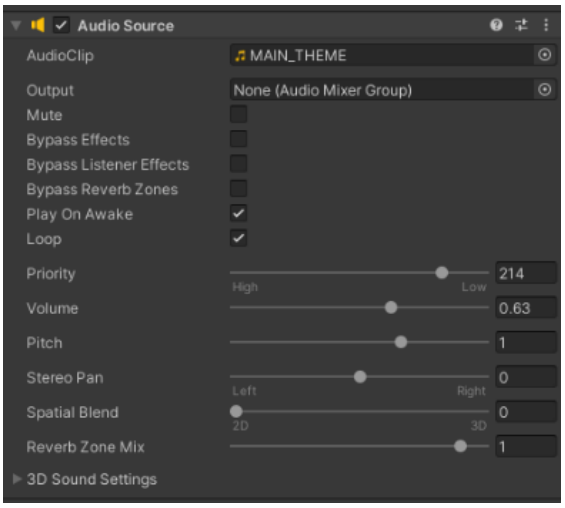

Figura 34. Componente *Audio Source* para la melodía principal.

Para el caso de la reproducción de sonidos en los minijuegos cuando se realiza alguna acción, es necesario añadir otro *Audio Source* aparte del anterior y sin añadir ningún clip. Es en los scripts de control de cada minijuego en donde es necesario crear dos o tres variables (según el número de audios que se reproduzcan) de los tipos: *AudioSource* y *AudioClip* teniendo que asignar en el inspector del script en cuestión la fuente a su correspondiente variable y los clips. En la figura 35 puede verse el componente *Audio Source* y el script de control con los clips llamados en el inspector.

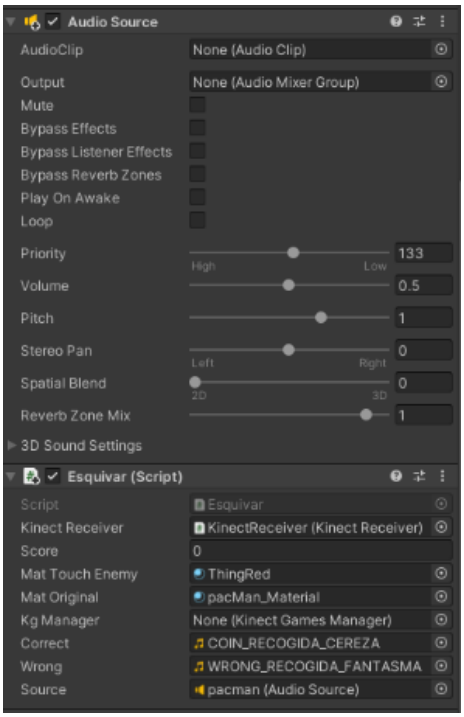

Figura 35. Componente *Audio Source* para los sonidos de correcto y mal para el juego *PacMan* junto con el script y la asociación de audios en este.

4.10 Modelaje y diseño del entorno

Como ya se ha mencionado, el entorno propuesto para este videojuego ha sido la habitación de un niño de una edad cercana a la del público que va dirigido. Para que el espacio no se sintiera vacío, se han añadido una serie de elementos que hacen que parezca una habitación más completa. Algunos de ellos han sido extraídos de modelos gratuitos en páginas como *Unity Asset store*, y otros han sido modelados directamente con las características que se necesitaban en el programa de modelaje *Blender.*

Algunos de los elementos que son originales del juego y creados para este son los que se observan en la figura 36.

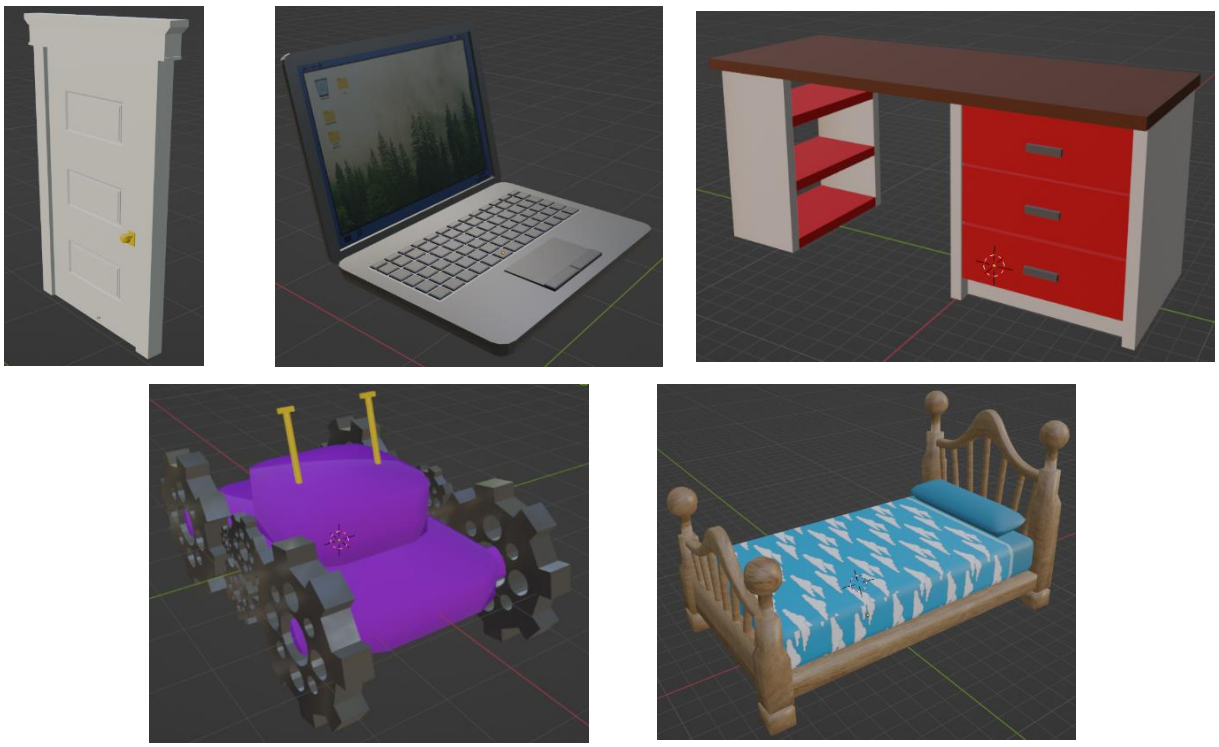

Figura 36. Elementos creados en Blender para el espacio del juego

Para la creación de estos, se comienza con la creación de un objeto simple como un cubo o un cilindro, a partir del cual, añadiendo caras, vértices o aristas se iban componiendo los objetos completos. Los principales comandos que se han utilizado para la composición de los objetos se representan en la tabla 2.

Tabla 2. Comandos principales de Blender para la creación de objetos.

Comando	Acción
Shift+A	Añadir un objeto
S + (x,y,z)	Escalar un objeto en una dirección o en todas
G + (x,y,z)	Desplazar un objeto en una dirección o en todas
R + (x,y,z)	Girar un objeto en una dirección o en todas
E	Extruir una cara o arista
Ctrl + R	Añadir un *loopcut* (con rueda del ratón se añaden cortes)
Ctrl + L	Selección de todas las caras/aristas/vértices de un objeto
A	Seleccionar todos los objetos visibles
Alt + click derecho	Seleccionar toda una línea de aristas
Ctrl + B	*Bevel* (suavizar bordes, con rueda del ratón se añaden cortes)
Mayus + D + (x,y,z)	Duplicar objeto y desplazarlo en un eje

Una vez con los objetos totalmente creados es necesario crear un material para cada uno y una textura que lo rellene. En el caso de los objetos creados, son texturas propias creadas para

el proyecto menos la textura de madera de la cama y estanterías [35], la del suelo [36] y la de las cortinas [37]. Los que parecen más complejos como la sábana de la cama, han sido dibujados primero como una imagen en el software *Photoshop*, y posteriormente aplicados a la textura.

Los objetos que han sido obtenidos de páginas externas pueden encontrarse en los enlaces de las siguientes citaciones [38-47].

Para el caso de la casa que se monta en el juego de Chop a House, primero fue creado el objeto en Blender con sus texturas y materiales y posteriormente se le dotó de una animación específica. Fueron añadidos una serie de huesos para facilitar su movilidad y a través de la ventana *Animation* del Software se fueron modificando su posición y rotación. En la figura 37 puede verse los huesos que fueron creados para realizar el movimiento.

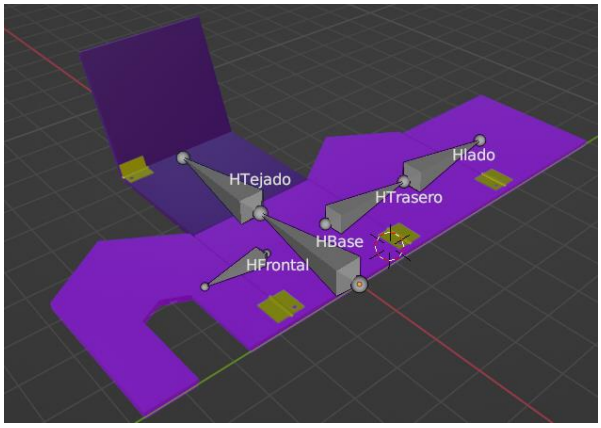

Figura 37. Huesos creados para la animación de la casa montándose.

En un inicio la casa está desmontada en el suelo. Para dotarla de movimiento y que se levantara, se pusieron dos puntos clave, uno en medio y otro al final. El del medio ponía todos los huesos y sus caras asociadas en pie y el del final modifica la posición y rotación de cada pared para que se ajuste perfectamente. En la figura 38 puede verse lo mencionado.

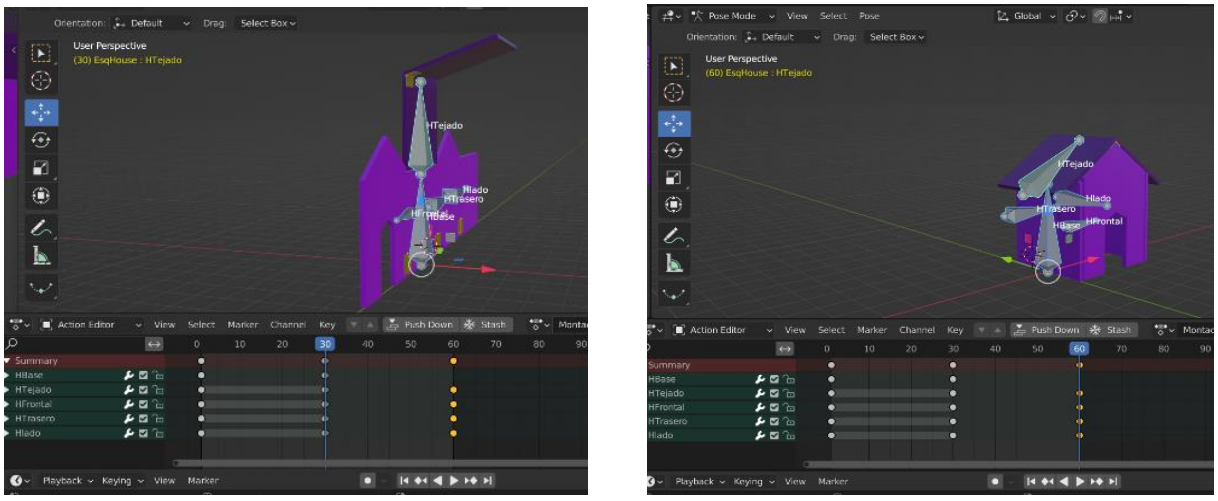

Figura 38. Inicio y fin de la animación de la casa montándose en la ventana Animation de Blender.

A la hora de asociar esta animación en Unity, importa a la escena y es necesario crear con controlador para la animación. En este caso se ha creado el controlador que se observa en la figura 39, en el que se controla a través del parámetro “animar”, el cual, si pasa a ser true, comienza la animación creada. Esto se controla desde el script ya mencionado *HouseAnim.cs.*

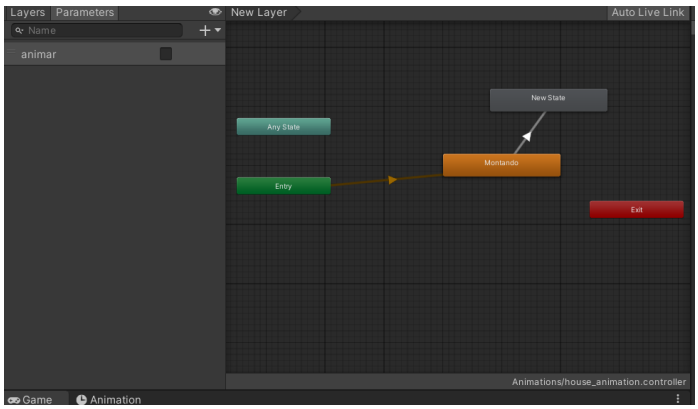

Figura 39. Controlador de la animación de la casa en el inspector *Animation* de Unity.

5. Resultados

Siendo el principal cometido de este proyecto la creación de un juego que sirva de base para una futura implementación de algoritmos inteligentes, a lo largo del proceso de creación se fueron aplicando pruebas para que, en su conjunto final, pueda funcionar lo más completamente posible.

Lo primero fue establecer el tamaño del entorno y comprobar que, a través del algoritmo de seguimiento de puntos con *NavMesh,* el tiempo de estos recorridos no es demasiado corto ni muy largo. Esto es importante para que, a la hora de establecer un tiempo de descanso entre minijuegos, queden resultados razonables y los usuarios no se fatiguen por tiempos cortos ni se aburran por tiempos largos. Se ha diseñado de tal forma que se presenta una habitación con objetos proporcionados al tamaño del entorno.

A la hora de situar estos puntos, se comprobó reiteradamente que el personaje no se paraba en ningún punto del recorrido, ya que, a la hora de calcular la malla del entorno, se generaban espacios en los que no se podía navegar y era necesario calcularla de nuevo. Tras muchas pruebas, se determinó que la malla del entorno está correctamente calculada y que éste es navegable sin huecos.

Por otro lado, a cada minijuego que se implementaba, era necesario comprobar su funcionamiento realizando el mismo proceso que haría un usuario del juego. Para ello, fue proporcionada una cámara *Kinect v2* para poder probarlo desde casa. Esta se usó para pruebas de cambios mínimos ya que, las pruebas de mayor calibre se realizaron con el equipo del CITSEM al poseer una tarjeta gráfica con mayor calidad. Así, se determinó el correcto funcionamiento de todas las partes del videojuego.

En relación con la transferencia y envío de datos a través de la web, se tuvieron una serie de problemas. Para la descarga de parámetros primero se probó la asignación sin un usuario introducido, es decir, utilizando los parámetros por defecto. Cuando esto fue aprobado, se procedió a la introducción de un usuario en el middleware para bajar los datos configurados en la web. Estas dos funcionalidades resultaron funcionar a la perfección en todos los casos. A la hora del envío de resultados, se consiguió visualizarlos en la web *Blexer-med*, dando a entender que se envían de forma correcta. El problema se encuentra en la visualización de algunos parámetros que, en la web, fueron definidos en su momento específicamente para el juego "*Phiby's Adventures 3D*" y que en este juego no se muestran correctamente. Esto se debe a la estructura interna de la web, la cual es necesaria modificar en un futuro para que sea accesible para todos los juegos.

Como último paso, sería necesario realizar pruebas con niños con movilidad reducida en centros o colegios. Así, se podría determinar qué elementos se deberían modificar y cuáles son los parámetros más adecuados para cada persona. A la hora de la implementación de IA en este proyecto, estas pruebas serían cruciales para entrenar a estos algoritmos.

6. Impacto del proyecto

El desarrollo de *Phiby's Toyland Escape* no solo intenta mejora la salud y el bienestar de los pacientes, sino que también pretende avanzar en la tecnología, preservar el medio ambiente y crecer económicamente. Este proyecto intenta convertir la rehabilitación en una experiencia más atrayente y motivadora. Esto es esencial para aumentar el compromiso al tratamiento y, por lo tanto, los resultados terapéuticos. Se espera que al hacer que la rehabilitación sea más divertida, los pacientes estén más dispuestos a participar activamente en la rehabilitación, lo que puede acelerar su recuperación y mejorar su calidad de vida. Además, este proyecto fomenta la inclusión social al permitir que las personas con discapacidades participen en actividades lúdicas y terapéuticas que de otro modo podrían estar fuera de su alcance.

Tecnológicamente, este juego impulsa el avance en el uso de inteligencia artificial y realidad virtual en el campo de la salud. Estas tecnologías no solo permiten una mayor personalización y eficacia de los tratamientos, sino que también promueven el desarrollo de nuevas herramientas y metodologías para la rehabilitación.

Desde una perspectiva ambiental, el desarrollo de videojuegos terapéuticos puede tener un impacto positivo al reducir la necesidad de desplazamientos frecuentes a centros de rehabilitación. Por otro lado, económicamente, al permitir la rehabilitación en el hogar y reducir la necesidad de sesiones presenciales, se pueden disminuir los costos asociados con el tratamiento continuo.

Por último, si se relaciona este proyecto con los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas, contribuye con varios, siendo los más importantes la mejora de la eficacia de los tratamientos de rehabilitación y hacer que sean más accesibles y atractivos para los pacientes (ODS 3: Salud y Bienestar) o la innovación en el sector de la salud (ODS 9: Industria, Innovación e Infraestructura).

El balance general del impacto social, de salud, ambiental, económico y tecnológicos que supone la creación de este proyecto es claramente favorable.

8. Conclusiones

8.1 Conclusiones

Tras la realización de este proyecto de fin de grado se concluye haciendo un resumen del trabajo realizado, los objetivos cumplidos y resultados obtenidos.

Este proyecto se centró en el desarrollo de un videojuego terapéutico para mejorar la experiencia de rehabilitación de pacientes mediante la gamificación de ejercicios físicos. Además de esto, debía servir como base para una futura implementación de algoritmos inteligentes. Esto surge tras la necesidad de cambio en elementos internos del juego para hacer de este algo que despertase curiosidad y adaptase mejor los parámetros pertinentes a cada paciente.

Así, se creó el videojuego "*Phiby's Toyland Escape*", el cual se basa en un entorno no navegable en el que se van sucediendo una serie de minijuegos. Para que fuera atractivo y entretenido para niños, se desarrolló un entorno basado en la habitación de un niño con colores vivos y diversos elementos que hacen de él algo más vistoso.

Para esta implementación, hubo una fase de familiarización con el juego en el que se basa, "*Phiby's Adventures 3D*", y tras esto y varias reuniones con la tutora, se establecieron los objetivos principales a cumplir.

Para abordar el objetivo de la adaptación del juego al paciente, se añadió un apartado de calibración para ajustar el esqueleto Kinect a cada usuario. Además, el juego está conectado con la web *Blexer-med* a través del middleware K2UM para la configuración de los parámetros pertinentes en cada jugador y la visualización de resultados.

En relación con la conservación de los movimientos originales del juego base, se plantearon cuatro minijuegos que mantienen los ejercicios de trepar, cortar, remar y esquivar. Estos se corresponden cada uno con una escena independiente en el entorno. Además, para que el jugador no pueda realizar una exploración del entorno y su movimiento entre minijuegos esté preestablecido, se planteó un sistema de puntos que el personaje recorre a través de la funcionalidad que sostiene Unity *NavMesh.* Estos espacios entre minijuegos son los tiempos configurables que debe haber para que el jugador no se fatigue.

Por último, se ha mantenido una estructura fácil de configurar para su futura modificación.

8.1 Trabajos futuros

Al ser un proyecto inicial en el que se pretende seguir trabajando, se muestran a continuación una serie de líneas de trabajo que pueden servir para el progreso de este proyecto.

- **Narrativa**: Idear una historia que se desarrolle entre los minijuegos, proporcionando un contexto y una motivación para las acciones del jugador. Por ejemplo, explicar por qué el personaje necesita energía, por qué debe subir a una estantería para recoger algo, y qué obtiene a cambio de completar los minijuegos aparte de conseguir energía.

- **Sistema de Recompensas**: Reemplazar el concepto de energía con otro tipo de recompensa, como las setas en "*Phiby's Adventures 3D*", que se utilizan para recargar la barra de energía. Esto añade un elemento de colección y recompensa que puede hacer el juego más atractivo.
- **Cinemáticas**: Incluir cinemáticas que desarrollen la historia, no solo al principio y al final del juego, sino también entre los ejercicios en la escena principal. Estas podrían incluir conversaciones con *NPCs* o monólogos internos del personaje.
- **NPCs**: Añadir *NPCs* en el entorno principal para interactuar con el personaje del jugador.
- Arreglar error de cámara al pasar desde arriba de la estantería al minijuego "*Chop a House*"
- **Mejorar minijuego "*Pull*"**: Mejorar las animaciones del juego, como hacer que las ruedas y la palanca del coche en el que está montado Phiby se muevan de manera realista cuando se desplaza. Además, cambiar el objeto que se muestra en la carretera como oponente por un NPC que compita con el personaje principal.
- **Mejorar minijuego "*Chop a House*":** añadir la posibilidad de realizar el movimiento con la mano izquierda, ya que actualmente solo es posible realzarlo con la derecha.
- **Interfaz Gráfica**: Desarrollar la parte gráfica del menú, incluyendo fondos y figuras en la escena de calibración que indiquen los movimientos a realizar además del logo del juego.
- **Calibración del Usuario**: Aplicar la calibración calculada al esqueleto real de la Kinect, ya que actualmente solo se calcula, pero no se aplica al usuario.
- **Decoración del Entorno**: Rellenar el entorno con decoración, muebles y otros elementos que hagan el escenario más atractivo y realista.
- **Dinámica de Pérdida**: Idear un sistema que permita que el jugador pueda perder en los minijuegos, como retroceder o perder energía, lo que añadiría un nivel de desafío y evitaría la sensación de progreso constante sin riesgo de fracaso.
- **Opciones de Menú en Juego**: Implementar una opción para que el jugador pueda levantar una de las dos manos y acceder a opciones como salir del juego, ver un mapa, revisar lo que queda en la escena, o ver los objetos recogidos.
- **Web *Blexer-med***: adaptación de la web para la correcta visualización de los resultados.

A la hora de incorporar los algoritmos inteligentes, se propone el desarrollo de nuevas escenas que utilicen los mismos movimientos, pero en diferentes puntos del juego. Esto permitirá que el la IA pueda decidir que escena mostrar en cada punto. Además, se plantea la creación de nuevos recorridos en la escena principal que se conecten con las escenas secundarias o que alternen con las ya existentes. Esta flexibilidad en los caminos permitirá que, en el futuro, la IA pueda decidir qué mostrar al jugador en función de su progreso y necesidades, ofreciendo una experiencia personalizada y adaptativa.

9. Referencias

[1] «Game Accessibility – IGDA». Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://igda.org/sigs/game-accessibility/

[2] C. Mangiron Hevia, «Accesibilidad a los videojuegos: estado actual y perspectivas futuras», *TRANS*, n.º 15, pp. 39-51, dic. 2011, doi: 10.24310/TRANS.2011.v0i15.3195.

[3] C. V. Maani *et al.*, «Virtual Reality Pain Control During Burn Wound Debridement of Combat-Related Burn Injuries Using Robot-Like Arm Mounted VR Goggles», *Journal of Trauma and Acute Care Surgery*, vol. 71, n.º 1, p. S125, jul. 2011, doi: 10.1097/TA.0b013e31822192e2.

[4] Dra. M. J. Busto Martínez y J. Pérez Martín, «Uso de los videojuegos en el tratamiento contra el dolor», sep. 2012, [En línea]. Disponible en: https://www.injuve.es/sites/default/files/Revista98_7.pdf

[5] «History», Hopelab. Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://hopelab.org/history/

[6] LANR, «Laboratorio de Investigación y Desarrollo de Aplicaciones Interactivas para la Neuro-Rehabilitación». Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://lanr.ifc.unam.mx/

[7] LANR, «Laboratorio de Investigación y Desarrollo de Aplicaciones Interactivas para la Neuro-Rehabilitación, "Juegos"». Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://lanr.ifc.unam.mx/juegos.html

[8] «Wii Fit», Nintendo of Europe AG. Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://www.nintendo.com/es-es/Juegos/Wii/Wii-Fit-283894.html

[9] «MIRA rehab», Zorginnovatie. Accedido: 10 de junio de 2024. [En línea]. Disponible en: https://zorginnovatie.nl/innovaties/mira-rehab

[10] «Interfaces Naturales para Aplicaciones Terapéuticas». Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://naturalinterfaces.etsist.upm.es/lineas-de-investigacion/body-group

[11] «Blexermed». Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://blexer-med.citsem.upm.es/index.php

[12] M. Jiménez Ramos, *Plataforma médica para el entorno de videojuego terapéutico «BLEXER»*, Proyecto de Fin de grado. Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2017.

[13] A. Aguilar Lopez, *Implementación de medidas de seguridad en plataforma médica para entorno de videojuegos terapéuticos.*, Proyecto de Fin de grado. Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2019.

[14] C. L. Vela, *Diseño e implementación de un entorno virtual de ejercicios físicos, basados en captura de movimiento*, Proyecto de Fin de Grado. Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2018.

[15] «Interfaces Naturales para Aplicaciones Terapéuticas». Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://naturalinterfaces.etsist.upm.es/sobre-nosotros/estudiantes-en-practicas/filip-racki

[16] «Interfaces Naturales para Aplicaciones Terapéuticas». Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://naturalinterfaces.etsist.upm.es/lineas-de-investigacion/body-group/sistema-blexer-v1

[17] *Interfaces Naturales para Aplicaciones Terapéuticas*. Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://naturalinterfaces.etsist.upm.es/lineas-de-investigacion/body-group/phibys-adventures-3d

[18] H. Qin, *Diseño e implementación de la historia, las mecánicas y el flujo del videojuego serio "Phiby's Adventures v2"*, Proyecto de Fin de grado. Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2020.

[19] J. I. Fuentes-Pila Martín, *Mejora de la lógica e incorporación de sistemas para el videojuego terapéutico Phiby's Adventure 3D.*, Proyecto de Fin de grado. Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2023.

[20] «Plataforma de desarrollo en tiempo real de Unity | Motor de 3D, 2D, VR y AR», Unity. Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://unity.com/

[21] «Visual Studio: IDE y Editor de código para desarrolladores de software y Teams», Visual Studio. Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://visualstudio.microsoft.com/es/

[22] «La evolución de Kinect y la importancia real de Microsoft Research». Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://www.xatakawindows.com/xbox/la-evolucion-de-kinect-y-la-importancia-de-microsoft-research

[23] B. Foundation, «About», blender.org. Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://www.blender.org/about/

[24] E. Botija Santamaría, *BlexerGameBase Asset*, Informe de prácticas. Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2022.

[25] «Freesound - Soft Game Theme Loop.wav by Mrthenoronha». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/Mrthenoronha/sounds/521656/

[26] «Freesound - Videogame Menu BUTTON CLICK by Christopherderp». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/Christopherderp/sounds/342200/

[27] «Freesound - Cartoon Game Theme Loop.wav by Mrthenoronha». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/Mrthenoronha/sounds/369920/

[28] «Freesound - SFX Magic by renatalmar». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/renatalmar/sounds/264981/

[29] «Freesound - Game Sound Wrong.wav by Bertrof». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/Bertrof/sounds/131657/

[30] «Freesound - Axe Cut #2 by birdswkaren». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/birdswkaren/sounds/712096/

[31] «Freesound - toy_engine_on_run_off_short_01.wav by magedu». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/magedu/sounds/451227/

[32] «Freesound - 8 Bit Adventure Theme (intro) by TheLastOneOnEarth». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/TheLastOneOnEarth/sounds/732429/

[33] «Freesound - 8-Bit Coin by TheDweebMan». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/TheDweebMan/sounds/277215/

[34] «Freesound - Invalid Selection by Beetlemuse». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://freesound.org/people/Beetlemuse/sounds/529384/

[35] R. Tuytel, «Plywood Texture • Poly Haven», Poly Haven. Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://polyhaven.com/a/plywood

[36] D. B. Baglioni Charlotte, «Laminate Floor 03 Texture • Poly Haven», Poly Haven. Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://polyhaven.com/a/laminate_floor_03

[37] R. Tuytel, «Fabric Pattern 05 Texture • Poly Haven», Poly Haven. Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://polyhaven.com/a/fabric_pattern_05

[38] «Board Skateboard 3D Model - TurboSquid 1274602». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/board-skateboard-3d-model-1274602

[39] «Free Backpack Punk Model - TurboSquid 1583590». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/backpack-punk-model-1583590

[40] «3D Ball Basket Basketball - TurboSquid 1205436». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/3d-ball-basket-basketball-1205436

[41] «Free Puzzle Mat 3D Model - TurboSquid 1195442». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/puzzle-mat-3d-model-1195442

[42] «Free 3D Nightstand Model - TurboSquid 2160004». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/3d-nightstand-model-2160004

[43] «Office 3D - TurboSquid 1710194». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/office-3d-1710194

[44] «Low Poly Car Vehicle Pack | 3D Land | Unity Asset Store». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://assetstore.unity.com/packages/3d/vehicles/land/low-poly-car-vehicle-pack-259182

[45] «Free Rubik Cube 3D - TurboSquid 1801670». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/rubik-cube-3d-1801670

[46] «Free 3D Shapespark Low Poly Curtains Kit - TurboSquid 1947311». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/3d-shapespark-low-poly-curtains-kit-1947311

[47] «Window 3d 3ds». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.turbosquid.com/3d-models/window-3d-3ds/1066625

10. Bibliografía

F. González, «La UNAM crea videojuegos para rehabilitar a pacientes con discapacidad motriz», WIRED. Accedido: 14 de mayo de 2024. [En línea]. Disponible en: https://es.wired.com/articulos/con-videojuegos-la-unam-quiere-mejorar-la-rehabilitacion-de-pacientes-con-discapacidad-motriz

L. Álvaro Gil, Mejora de la experiencia y del contenido del videojuego terapéutico "Phiby's Adventures 3D", Proyecto de Fin de grado. Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2020.

L. Molero Salazar, Diseño e implementación del entorno para el juego serio terapéutico "Phibys Adventures v2", Proyecto de Fin de grado. Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, 2019.

Anexo I: Presupuesto

En este anexo se evalúan todos los recursos empleados en la realización de este proyecto, incluyendo materiales, personal y licencias de software.

Mano de obra:

- 4 meses para el desarrollo del proyecto→285 horas
- 2 meses para la redacción del informe y pruebas pertinentes→40 horas

En total se estima un tiempo de trabajo de 325 horas. Tomando como referencia el salario medio de una Ingeniera recién titulada [1] el cual ronda unos 20.600 €/año, considerando las 40 horas de trabajo de una jornada normal, se tendrían 10.7 €/hora. Por lo tanto, el coste de personal de este proyecto ha supuesto 3.477,5 €.

Material y licencias:

El principal motor de desarrollo de este proyecto ha sido el ordenador, además de los softwares, la mayoría con licencias gratuitas y la cámara *Kinect One* de *Microsoft*. Se desgrana abajo los componentes del equipo informático.

- Portatil→749 € [2]
    - Intel ® Core™ i7-1165G7 11th Gen 2.80 GHz
    - Memoria RAM 8 GB
    - 500 GB Disco Duro 5400 rpm
    - Intel Graphics
    - Sistema operativo *Windows* 10 *Home*
- Periféricos del ordenador (ratón, teclado, monitor extra) → 150 € [3] [4] [5].
- Cámara *Kinect* v2 de *Xbox One* → 144.95 € [6]
- Adaptador para *Microsoft Kinect* V2 para *Windows* 10 PC → 29.98 € [7]
- *Software Unity, Blender, Visual Studio 2019* → licencias gratuitas.

El precio de los materiales mencionados está sujeto a cambios a lo largo del tiempo según el contexto económico del momento.

Coste total

Con lo anterior mencionado, se obtiene un coste total de 4551.43 €, desglosado en la tabla 3.

Tabla 3. Presupuesto desglosado del coste humano y material del proyecto

Descripción	Unidades	Coste parcial (€)
Ingeniero junior (325 h)	1	3.477,5
Portátil	1	749
Periféricos	1	150
Cámara Kinect v2	1	144,98
Adaptador Kinect-PC	1	29,98
Licencias software	1	0
Coste total		4.551,43

A.1 Referencias

[1] «¿Cuánto Cobra un Ingeniero de Telecomunicaciones? (Sueldo 2024) | Jobted.es». Accedido: 19 de junio de 2024. [En línea]. Disponible en: https://www.jobted.es/salario/ingeniero-telecomunicaciones

[2] «Portátil | Lenovo IdeaPad 5 15ITL05, 15.6" Full HD, Intel® Core™ i7-1165G7, 8GB RAM, 512GB SSD, Intel® Iris® Xe Graphics, Sin sistema operativo», MediaMarkt. Accedido: 19 de junio de 2024. [En línea]. Disponible en: https://www.mediamarkt.es/es/product/_portatil-lenovo-ideapad-5-15itl05-156-full-hd-intelr-coretm-i7-1165g7-8gb-ram-512gb-ssd-intel-irisr-xe-graphics-sin-sistema-operativo-1531637.html

[3] «Logitech K120 Teclado USB Negro | PcComponentes.com». Accedido: 19 de junio de 2024. [En línea]. Disponible en: https://www.pccomponentes.com/logitech-k120-teclado-usb-negro

[4] «Logitech M90 Ratón Óptico 1000DPI | PcComponentes.com». Accedido: 19 de junio de 2024. [En línea]. Disponible en: https://www.pccomponentes.com/logitech-m90-raton-optico-1000dpi

[5] «Samsung Essential S24C330GAU 24" LCD IPS FullHD 100Hz FreeSync | PcComponentes.com». Accedido: 19 de junio de 2024. [En línea]. Disponible en: https://www.pccomponentes.com/samsung-essential-s24c330gau-24-lcd-ips-fullhd-100hz-freesync

[6] «Kinect Xbox One. Xbox One: GAME.es». Accedido: 19 de junio de 2024. [En línea]. Disponible en: https://www.game.es/kinect-xbox-one-xbox-one-102796

[7] «Kinect - Adaptador para Xbox One S/X y Windows 8/8.1/10 PC, Microsoft Kinect 2.0 Sensor fuente de alimentación con enchufe europeo : Amazon.es: Videojuegos». Accedido: 19 de junio de 2024. [En línea]. Disponible en: https://www.amazon.es/Kinect-Adaptador-Windows-Microsoft-alimentaci%C3%B3n/dp/B0BHT4NT22/ref=asc_df_B0BHT4NT22/?tag=googshopes-21&linkCode=df0&hvadid=646887993278&hvpos=&hvnetw=g&hvrand=2106398332315468156&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=20292&hvtargid=pla-1928531025034&psc=1&mcid=9afa6042cd623f2da085408232621e97

Anexo II: Manual de usuario

En este anexo se explica el proceso por parte del usuario para interactuar con el juego.

En este proyecto se ha entregado un archivo que contiene el ejecutable del juego, el middleware y los códigos fuente. Tras descomprimirlo, para iniciar el middleware se debe entrar en la carpeta *MiddlewareBlexer,* iniciar la solución de *Visual Studio* llamada "Kinect Middleware" y darle a iniciar en la ventana de *Visual Studio*. Queda pendiente crear un ejecutable del middleware. Para iniciar el juego, se debe abrir el ejecutable llamado "Phibys_Toyland_Escape_Julio2024".

Al abrir el ejecutable del juego, si no se ha iniciado sesión en el middleware, se pedirá un nombre de usuario para dar nombre al archivo de resultados, si se desea, se puede dar a continuar sin introducir este. En este caso, se jugaría con los parámetros por defecto. Tras esto, aparecerá el menú principal del juego en el que se muestran las opciones *Play, Minigames, Options y Controls,* como puede verse en la figura.

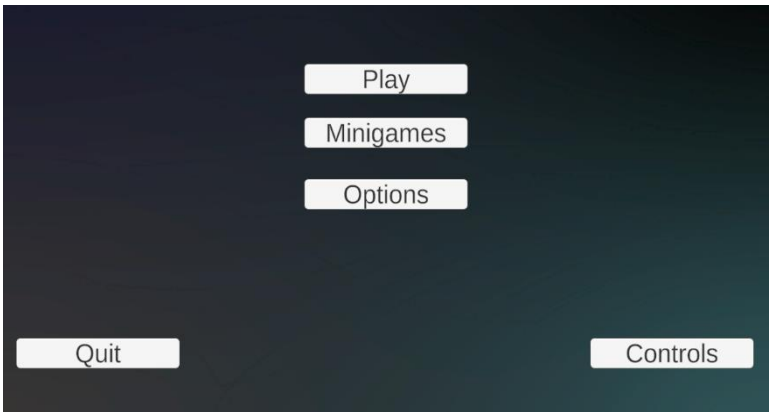

Figura 40. Menú principal del juego *Phiby's Toyland Escape.*

Para iniciar el juego en su historia principal, es necesario darle a *Play.* Para iniciar el recorrido del personaje, hay que levantar una mano hasta que la nube superior que aparece se vuelva verde y bajar el brazo. A partir de aquí, *Phiby* va trasladándose a los diferentes puntos en donde saltan las escenas de los minijuegos hasta llegar al final. Si se selecciona en el menú principal la opción *Minigames*, aparecerá una ventana en la que se puede seleccionar el minijuego al que se quiere acceder, sin pasar por la escena principal. Si en algún momento del juego se quiere salir al menú principal, basta con darle a la tecla "*esc"*.

Si se quiere calibrar el juego, hay que entrar en el menú de *Options* y darle a *Calibrate.* Si no está la Kinect conectada o el middleware iniciado, esta última opción de calibrar no funcionará.

En este menú también se encuentra *History* que muestra la historia principal del juego, y los créditos del proyecto.

Para consultar los principales controles del juego, se puede consultar en el menú *Controls.*

Para salir del juego, desde el menú principal se debe dar a la opción *Quit.*