TELECOMUNICACIÓN

Campus Sur

POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Desarrollo e implementación de middleware entre Blender, Kinect y otros dispositivos

AUTOR: Ignacio Gómez-Martinho González

TUTOR (o Director en su caso): Martina Eckert

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

DEPARTAMENTO: Departamento de Teoría de la Señal y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Mª Luisa Martín Ruiz

VOCAL: Martina Eckert

SECRETARIO: Enrique Rendón Angulo

Fecha de lectura: 22 de julio de 2016

Calificación:

El Secretario,

Resumen

El objetivo de este proyecto es desarrollar un programa que sirva de intermediario (*middleware*) entre una cámara de captura de movimiento y un entorno de desarrollo de videojuegos, permitiendo la creación de juegos controlados por movimientos corporales, o *exergames*. El objetivo a largo plazo es el desarrollo de juegos adaptativos destinados a usuarios con movilidad reducida, especialmente a niños y adolescentes, que combinen el entretenimiento con el ejercicio físico necesario para la rehabilitación de pacientes. El sistema no sólo debe leer los movimientos corporales del jugador y trasladarlos a los controles del videojuego, sino también reconocer las limitaciones físicas de cada usuario particular, adaptando los movimientos requeridos por el juego a las necesidades del paciente.

Este proyecto se basa en el uso de la cámara de control de movimiento Kinect, desarrollada por Microsoft. Este sensor permite reconocer la posición en el espacio y la orientación de hasta veinte puntos, o articulaciones, del cuerpo del usuario. Utilizando esta información, es posible reconstruir en un entorno 3D una simulación esquematizada del esqueleto del jugador, que imitará todos sus movimientos en tiempo real. El software elegido para el desarrollo de los videojuegos es el Blender, una solución gratuita de código abierto que permite el modelado y programación de juegos en tres dimensiones, con cualquier grado de complejidad.

Para poder integrar en Blender los movimientos registrados por la cámara, el sistema está dividido en dos componentes: el principal es una aplicación autónoma que accede regularmente a la interfaz de la Kinect y obtiene los datos de posición del usuario. Mediante su propio protocolo de comunicación, la aplicación transmite esos datos al segundo componente: un receptor, cuyo código se incorpora y ejecuta en las aplicaciones desarrolladas con Blender. Este receptor interpreta los datos recibidos y reconstruye el esqueleto del jugador en tiempo real, permitiendo su incorporación al sistema de control del juego. Los movimientos del jugador pueden utilizarse de dos formas: por una parte, haciendo que el personaje protagonista los imite, y por la otra, midiendo la velocidad de los movimientos o la distancia entre determinados puntos del cuerpo, y activando determinados comandos en función de los valores obtenidos.

El receptor, además, puede guardar en la memoria permanente del ordenador la anatomía y la capacidad de movimiento de cada usuario, y utilizar estos datos para amplificar los movimientos del esqueleto. Esto permite que ligeros gestos, realizados por jugadores con movilidad reducida, sean interpretados por el juego como movimientos tan amplios como los que pueda realizar cualquiera.

El middleware entre Blender y Kinect se ha integrado en un programa contenedor, el Chiro, con capacidad para agrupar una serie de *middlewares* que comuniquen Blender con distintos dispositivos de interfaz natural. Se han implementado uno para *smartphones* y otro para cascos de realidad virtual, así como las herramientas necesarias para desarrollar otros similares en el futuro.

En la última fase del proyecto se ha iniciado el diseño de un juego de aventuras 3D con funciones de rehabilitación, al que se ha dado el nombre provisional de “Blexer” (Blender *Exergames*). De los que en el futuro formarán parte de él, se han implementado cinco ejercicios o minijuegos, en colaboración entre el autor y estudiantes en prácticas. Estos juegos han sido el centro de unas pruebas realizadas con dos grupos de personas, con y sin limitaciones motoras, con el objetivo de mostrar la utilidad y la funcionalidad del sistema. Los resultados han sido muy positivos.

En total, este trabajo dio lugar a dos publicaciones en conferencias: la ISCE 2015 en Madrid [1] y el ICCE 2016 en Berlín [2].

Abstract

The goal of this project is to develop a program that works as a middleware between a motion capture camera and a videogame development environment, allowing the development of videogames controlled by corporal movements, called exergames. The overall goal is the development of adaptable games targeted towards physically handicapped users, especially children and teenagers, combining entertainment with the physical exercise necessary for the patient's rehabilitation. The system must not only capture the player's body motion and use it as input for videogames, but also recognize the physical limitations for every individual user, adapting the required movements in the game to the patient's needs.

This project is based on the use of the motion control camera named Kinect, developed by Microsoft. This sensor allows to read the spatial location and the orientation of up to twenty points, or joints, in the user's body. Using this information, it is possible to rebuild a schematized simulation of the player's skeleton in a 3D environment, which will mimic his/her every move in real time. The software chosen for the development of the games is Blender, a free and open-source solution that allows the modeling and programming for tri-dimensional games with any level of complexity.

In order to integrate the movements registered by the camera into Blender, the system is divided into two components: the main one is a standalone application that accesses to the Kinect interface at regular intervals, and reads the user's position data. Using its own communication system, the application passes the data on to the second component: a receiver, which code is integrated and executed from the applications developed on Blender. This receiver reads into the received data and rebuilds the player's skeleton in real time, allowing its incorporation to the game's control system. The user's movements can be used in two ways: on one side, making the protagonist character mimic them, and on the other one, measuring the speed of the motion or the distance between determined points in the body, and activating certain body joints to activate game commands in function of the resulting values.

The receiver can also store in the computer's permanent memory each user's anatomy and ability for movement, and it can use these data to amplify the skeleton's motion. This allows for slight gestures, performed by handicapped users, to be read by the game as movements as wide as anyone's.

The middleware between Blender and Kinect has been integrated within a container program, named Chiro, with the ability to group an array of different middlewares communicating Blender with different Natural User Interface devices. A middleware for smartphones has been implemented, and another one for a virtual reality headgear, as well tools allowing the future development of other similar applications.

In the last phase of the project, design has been initiated on a 3D adventure game with exercises for rehabilitation, which has been given the working name of "Blexer" (Blender exergames). From among the ones that will be a part of it, five exercises or minigames have been implemented, in a collaboration between the author and students in practices. These

games have been the center of a series of tests performed with two groups of people, with and without motion handicaps, with the goal of proving the usefulness and functionality of the system. The results have been very positive.

Overall, this work gave rise to the following two conference publications: at ISCE 2015 in Madrid [1] and at ICCE 2016 in Berlin [2].

Índice de contenidos

Lista de acrónimos

ADB:	*Android Debug Bridge*, Puente de depurado de Android
API:	*Application Programming Interface*, interfaz de programación de aplicaciones
BGE:	Blender Game Engine
CITSEM:	Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad
HMD:	*Head-Mounted Display*, Visor adaptado a la cabeza
NUI:	*Natural User Interface*, interfaz natural de usuario
GRyS:	Grupo de Redes y Servicios de Próxima Generación
OSVR:	*Open Source Virtual Reality*, Realidad Virtual de Código Abierto
RA:	Realidad Aumentada
SDK:	*Software Development Kit*, kit de desarrollo de software
RGB:	*Red, Green, Blue*. Rojo, verde y azul.
TCP:	*Transmission Control Protocol*, Protocolo de Control de Transmisión
UDP:	*User Datagram Protocol*, protocolo del nivel de transporte basado en el intercambio de datagramas

1. Introducción

En los últimos años, el mundo de la informática y las telecomunicaciones ha visto un aumento en presencia y popularidad de las llamadas "Interfaces Naturales de Usuario" (NUI, *Natural User Interface*). Éstas sustituyen a las interfaces habituales de un sistema informático, como ratones, teclados o monitores, por otras más adaptadas al cuerpo humano: comandos por voz, detección de movimientos, reconocimiento facial, etc.

La creciente disponibilidad de estas tecnologías se vio reflejada también en el panorama de los videojuegos, donde a partir de 2006 se volvieron increíblemente populares los sistemas de control por movimiento. Desde los mandos con acelerómetro como el WiiMote de Nintendo [3] o el PlayStation Move de Sony [4], hasta las cámaras de detección corporal como la EyeToy, también de Sony, [5] y la Kinect de Microsoft [6].

Esta última, en concreto, ha llamado la atención de desarrolladores de software en ámbitos muy distintos a la industria del entretenimiento. Su función principal es detectar en tiempo real la posición espacial de los jugadores que se encuentren en su campo de visión y, mediante un escaneo del entorno en 3D, realizar una representación esquemática de sus cuerpos, a la que se conoce como "esqueleto".

Una de las líneas de trabajo abiertas en el CITSEM (Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad) [7] investiga la aplicación de las interfaces naturales en la ayuda de personas con movilidad reducida, utilizando el Kinect como herramienta principal. Además de sus posibilidades como sistema de control en programas ya existentes, se están explorando sus aplicaciones en ejercicios de rehabilitación.

Desde la aparición de los controles por movimiento en consolas, los juegos orientados a la realización de ejercicio físico (*exergames*) han sido una constante, hasta el punto de convertirse en la aplicación más popular de estos accesorios. También se les ha dado una gran cantidad de usos en el entorno médico, ya que permiten aumentar el atractivo visual de los ejercicios y mejorar el *feedback* sobre su realización. Sin embargo, casi todos estos juegos tienen una temática deportiva (están ambientados en gimnasios), y los que ofrecen otras excusas argumentales suelen tener esquemas de juego muy simples, basados en la repetición constante de una limitada serie de actividades.

El objetivo a largo plazo del CITSEM es desarrollar un videojuego que ofrezca una experiencia más amplia, emulando a los juegos comerciales de aventuras y de plataformas. Este juego deberá implementar un control que requiera del usuario la realización de varios movimientos físicos que imiten la situación del personaje principal: usar los pies para correr y saltar, las manos para realizar ataques o utilizar objetos, etc. Estos ejercicios deben estar integrados en la ambientación y la historia de forma que el jugador deje de percibirlos como un esfuerzo físico, y le motive a continuar su progresión en el juego por diversión en lugar de por recomendación médica. El juego, además, adaptarse al perfil y las capacidades del jugador, teniendo muy en cuenta el rango de movimientos que sea capaz de hacer. Por ejemplo, si fuera necesario, debería interpretar un ligero movimiento con la mano como una

gran sacudida del brazo. También debe considerar las necesidades terapéuticas del jugador para presentarle en cada momento los ejercicios que le correspondan para la rehabilitación, con el nivel de dificultad oportuno para ese momento.

Para poder alcanzar este objetivo a largo plazo, primero hay que desarrollar las herramientas de software necesarias. Esta fase es la que constituye el eje central de este PFG: la creación de un entorno que posibilite el desarrollo de aplicaciones con control por Kinect en Blender [8], un programa de modelado en 3D, animación y videojuegos.

Para hacer posible la compatibilidad entre Blender y Kinect serán necesarios dos elementos: uno será un código instalable en Blender, o *add-on*, que permita que sus juegos reciban los datos de posición proporcionados por la Kinect y los integren dentro del sistema de control. El otro será un programa intermedio, o *middleware*, que se ocupe del manejo de la Kinect mediante su entorno de desarrollo oficial (SDK), obtenga los datos de posición y se los transmitirá al juego.

El *add-on* proporcionará el entorno necesario para que futuros desarrolladores puedan utilizar el control por movimientos de forma intuitiva, incluyendo las herramientas que permitan integrar los movimientos en la lógica interna del juego y, sobre todo, el reconocimiento de usuarios y la amplificación de movimientos.

El middleware se diseñará con un enfoque modular, que permita añadir funcionalidades futuras de forma fácil. Si bien el centro de este proyecto es la Kinect, es posible partir de este sistema para añadir la compatibilidad con otros dispositivos de cualquier tipo: cualquier *smartphone*, por ejemplo, incorpora un acelerómetro que permite utilizarlo como detector de movimientos al estilo del WiiMote. También se contempla la posibilidad de establecer una comunicación entre Blender y cascos de realidad virtual, como el Oculus Rift [9], que realizan una detección de los movimientos de la cabeza para ofrecer al jugador una vista estereoscópica en primera persona, dando la sensación de estar físicamente dentro del mundo del juego. Otros sensores podrían utilizarse para monitorizar la actividad del jugador, por ejemplo un cardiómetro, y evaluar así su esfuerzo físico para que el juego reaccione y se ajuste de forma acorde.

2. Marco tecnológico del proyecto

Uno de los grupos de investigación del CITSEM, el GRyS (Grupo de Redes y Servicios de Próxima Generación), incluye en sus áreas de trabajo las aplicaciones de las tecnologías de realidad aumentada (RA). Estas tecnologías se caracterizan por integrar datos digitales y elementos del mundo físico, por ejemplo procesando los datos de una señal de vídeo para crear un modelo 3D de su contenido y hacer añadidos a la imagen. La realidad aumentada, la mayoría de las veces, utiliza Interfaces Naturales de Usuario.

Estas interfaces pueden ser de gran ayuda para usuarios con movilidad reducida, que tal vez tengan dificultades para manejar un ordenador con las interfaces clásicas. En lugar de ratón y teclado, por ejemplo, podrían controlarlo mediante leves gestos de la mano. Pero además de facilitar el manejo de aplicaciones existentes (un terreno cubierto por el proyecto MoKey [10]), estas interfaces se pueden utilizar en programas diseñados para la rehabilitación de pacientes.

Para esta tarea se ha elegido la Kinect, un sensor desarrollado por Microsoft capaz de reconocer los movimientos y la posición espacial del usuario, y que aparece representado en la Figura 1. Se comercializó inicialmente como accesorio para la consola Xbox 360, pero desde entonces Microsoft ha puesto su entorno de desarrollo a disposición de los usuarios [11]. Utiliza una cámara de vídeo RGB y una combinación de emisor y receptor de luz infrarroja para procesar su entorno en tres dimensiones, pudiendo reconstruir virtualmente en tiempo real la pose de los usuarios que estén en su campo de visión. Esto permite su utilización en muchos ámbitos, pero su aplicación más extendida con diferencia es la del control de videojuegos por movimiento corporal.

Figura 1 - Kinect para Xbox 360

Existen ya varios trabajos sobre el uso de la Kinect en proyectos de salud. H. M. Hondori y M. Khademi [12], por ejemplo, evalúan más de cuarenta sistemas de cuidado médico implementados con la Kinect. Éstos tienen propósitos de toda clase: entrenamiento del equilibrio, ayuda a la terapia, rehabilitación cognitiva etc. Pero muy pocos son de rehabilitación, y de entre ellos la mayoría están orientados a pacientes de ictus.

D. Webester y O. Celik [13] evalúan la utilidad de la Kinect en el cuidado de mayores y en sistemas de rehabilitación, también para pacientes con ictus. Encuentran varias limitaciones, especialmente en la precisión de la detección, pero en general lo consideran la mejor opción dentro de los sistemas orientados a los videojuegos, comparado concretamente con la cámara EyeToy o el mando WiiMote.

También pueden encontrarse varias aplicaciones ya desarrolladas que permiten hacer rehabilitación con la Kinect. Proyectos como PhysioMate [14], VirtualRehab [15], KinectoTherapy [16] o NeuMimic [17] ofrecen entornos virtuales para realizar las actividades de rehabilitación, pero no dejan de ser versiones electrónicas de programas de ejercicios estándar. El KineLabs [18], de Hong Kong, o el KinectUmbrella [19] de la Universidad Politécnica de Valencia ofrecen objetivos y ambientaciones más lúdicos, pero están muy orientados a minijuegos cortos, lo que en consolas se conoce como "party games" de dos minutos de duración o menos, sin ningún tipo de argumento ni progresión.

Blender es el programa que se ha elegido para el desarrollo de los videojuegos. Se trata de un software gratuito, que permite crear desde cero vídeos con calidad fotorrealista y aplicaciones interactivas de cualquier tipo. A diferencia de otros entornos de desarrollo en 3D, permite el libre modelado de objetos, que luego se integran en los escenarios junto con sistemas de luces, partículas, código ejecutable, etc.

Existen ya varias herramientas que permiten la conexión entre Blender y la Kinect, y cada una tiene un enfoque distinto con una funcionalidad en concreto. Están basadas en distintos entornos de desarrollo, es decir, distintas bibliotecas de programación. El más importante, por supuesto, es el kit de desarrollo oficial (SDK) distribuido por Microsoft, pero éste fue puesto a disposición del público general mucho después del lanzamiento del dispositivo. Antes de que eso ocurriera se crearon varias alternativas de código abierto, como el SDK de OpenNI (*Open Natural Interaction*) [20]. Ésta es una organización sin ánimo de lucro creada por varias empresas dedicadas a las interfaces naturales, y tiene kits de desarrollo tanto para Kinect como para otros dispositivos de captura de movimiento.

La desarrolladora Delicode utilizó el entorno de OpenNI para desarrollar su propia interfaz, el NI-Mate [21]. Se trata de un *middleware* que lee los datos de la Kinect y los retransmite utilizando el protocolo OSC o el MIDI, a elección del usuario. NI-Mate es compatible con varios programas, no sólo con Blender, y para cada uno hay disponible un *add-on* que instala un módulo receptor. Este *add-on* permite recibir e introducir el movimiento del jugador en el entorno 3D de Blender, tanto durante la edición como en un juego ejecutándose. En ambos casos el *add-on* recibe en cada instante de tiempo los datos del usuario, y crea una serie de objetos en el entorno 3D que representan la posición de sus articulaciones.

El mayor inconveniente del NI-Mate es que se trata de un programa de pago, que exige la compra de una licencia. Además, el entorno OpenNI dejó de recibir soporte y actualizaciones a principios de 2014, una vez que el SDK oficial estuvo ya disponible a todos los desarrolladores.

De entre las soluciones gratuitas basadas en el SDK, cabe destacar también la *Blender Loop Station*, o Bloop [22]. Consiste en un *add-on* para Blender que permite controlar el movimiento de un modelo de personaje utilizando las manos del usuario, a la manera de un marionetista. Sin embargo, este sistema está orientado a la animación de personajes y la realización de vídeos, no al control de un videojuego.

3. Solución desarrollada

3.1. El *middleware*

A la hora de añadir funciones a Blender para hacerlo compatible con la Kinect, la opción más evidente a priori es escribir un código que, desde los ejecutables de los juegos, acceda a la interfaz de la Kinect y obtenga los datos sobre los movimientos del jugador, pero Blender no proporciona esta posibilidad. El software solo permite ejecutar código en el lenguaje Python [23], pero el kit de desarrollo oficial de Kinect es una biblioteca que sólo puede ser utilizada en los lenguajes C#, C++ y Visual Basic.

Esto hace necesaria la codificación de un software intermediario (*middleware*), en uno de los lenguajes compatibles, que controle el funcionamiento del sensor, obtenga sus datos y se los retransmita a Blender. La mayoría de los recursos online sobre el desarrollo con el SDK 1.8 de Kinect contemplan sólo el desarrollo en C#, por lo que éste es el lenguaje que se ha elegido. También es el lenguaje utilizado por KinectOSC [24], una aplicación de código abierto desarrollada por Andre Hayter que ha servido como modelo para este *middleware*, si bien sus opciones son limitadas y poco idóneas para el resultado que se quiere obtener.

Como se mencionó en la introducción, Kinect es sólo uno, el principal, de los dispositivos cuyo soporte se quiere añadir a Blender. Se ha codificado también un *middleware* para móviles con el sistema operativo Android y se ha integrado el software creado por José Zarco [25] para el Oculus Rift [9]. Para hacer el uso de estos programas más fácil para el usuario, se ha decidido que estos *middlewares* operan como módulos de un solo programa principal, al que se ha puesto el nombre de “Chiro”.

En la Figura 2 se ilustra el funcionamiento global de este sistema de comunicación: el Chiro agrupa los distintos *middlewares* codificados, cada uno comunicándose por USB con un tipo de dispositivo. Cada uno de ellos gestiona su comunicación con Blender, realizada a través de sus respectivos *add-ons*, que reciben los datos ofrecidos por cada sensor.

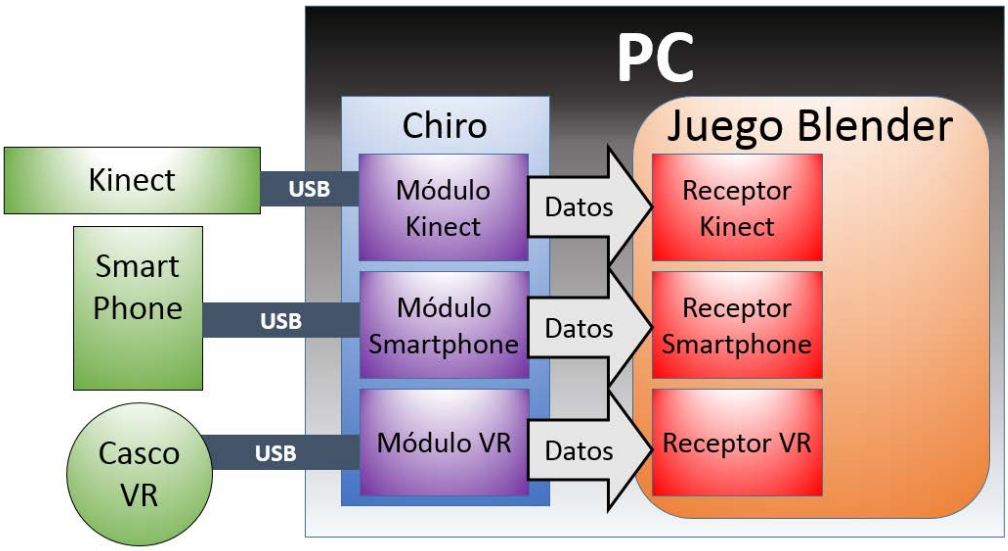

Figura 2 - Diagrama de bloques de la comunicación.

3.1.1. El contenedor Chiro

Conceptualmente, Chiro es un programa muy simple: no es ni más ni menos que una interfaz gráfica consistente en una ventana a la que va añadiendo distintas **pestañas**. Cada una de estas pestañas contiene el código y los controles de la comunicación entre Blender y un dispositivo específico de NUI.

Las pestañas pueden ser radicalmente diferentes entre sí a nivel de funcionamiento, e incluso pueden estar escritas en diferentes lenguajes. Lo verdaderamente importante es que su formato sea tal que Chiro pueda cargarlas desde un archivo externo y añadirlas a su interfaz. Actualmente Chiro no es capaz de interactuar con las pestañas de otra forma que no sea inicializarlas o destruirlas, por lo que éstas deberán funcionar de forma autónoma y contener en su interfaz todos los controles necesarios para que el usuario influya en la comunicación.

Al iniciar el programa Chiro, éste leerá de un archivo la **lista** de pestañas disponibles para añadir. Este archivo no contiene el código del *middleware*, sino un conjunto de objetos de la clase WindowReference (codificada para este proyecto), que contienen en sí toda la información necesaria para cargar un *middleware* en forma de pestaña: su nombre, su constructor, el archivo en el que está definido, etc. Para cada uno de estos objetos, Chiro intentará acceder a la **biblioteca** donde esté definida la pestaña correspondiente y cargarla en su interfaz. Si no es compatible, dará al usuario la opción de instalar las bibliotecas oportunas, ejecutando un archivo de instalación especificado en el objeto WindowReference. Si falta algún archivo, el usuario podrá buscarlo de forma manual. Si la carga tiene éxito, el *middleware* se incorporará al programa. Si por cualquiera de las razones anteriores no es así, se creará una pestaña vacía para indicar que ha habido un intento fallido, y se procederá a intentar cargar la siguiente pestaña de la lista. Todo este proceso se describe con más detalle en el Anexo 2.

3.1.2. KinectWindow

Esta clase, heredera del objeto gráfico TabPage (que representa una pestaña), constituye el middleware entre Blender y Kinect. Su función, por lo tanto, se divide en dos partes: guardar y actualizar la información sobre la pose de los jugadores, y transmitir esa información al juego programado en Blender. Ambas tareas pueden ejecutarse de formas diferentes en función de una serie de parámetros, por lo que esta pestaña ofrece una interfaz gráfica con controles que permiten al usuario modificarlos a su conveniencia. Esta interfaz se encuentra representada en la Figura 3, en una versión independiente no insertada en el Chiro. Para simplificar su uso, la pestaña inicializa sus parámetros con valores por defecto que permiten un funcionamiento correcto. Incluso se ha implementado un sistema que permite a los juegos configurar el *middleware* remotamente según sus necesidades, por lo que la interacción necesaria entre la interfaz y el usuario se limita a poner en marcha el Chiro.

Para entender el funcionamiento interno de la pestaña, será necesario explicar en detalle el conjunto de factores que influyen en las dos mitades del proceso, las especificaciones buscadas y su implementación.

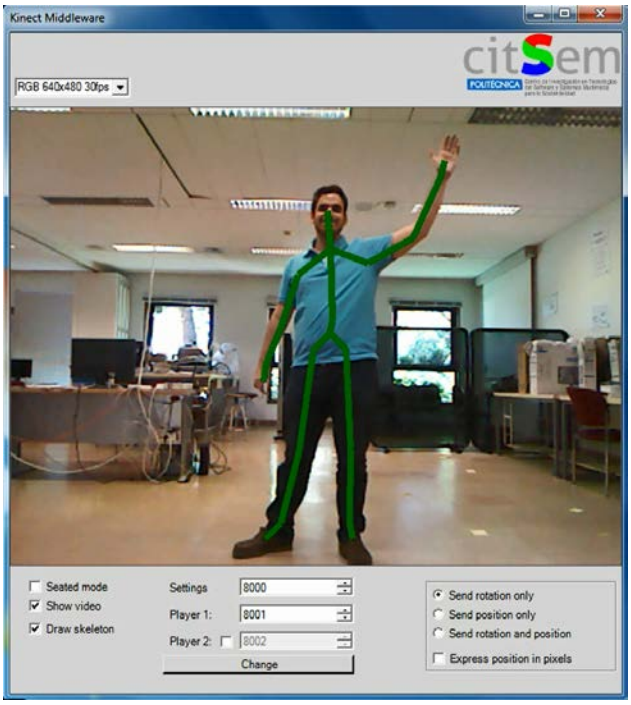

Figura 3 - Interfaz gráfica de la KinectWindow

3.1.2.1. La interacción con Kinect

La obtención de datos de esqueleto proporcionados por la Kinect, así como metadatos adicionales, se ha implementado mediante el uso del kit de desarrollo oficial proporcionado por Microsoft en su página web. Concretamente, se ha usado la versión 1.8 del *Kinect for Windows SDK* [26], su última actualización antes del lanzamiento de la cámara Kinect 2.0.

El objetivo de esta sección del middleware es almacenar en una variable o estructura de memoria temporal tanto la posición espacial de las articulaciones del usuario como las rotaciones de sus huesos. Toda esta información se encuentra contenida en la clase Kinect.Skeleton, definida en el SDK. Para poder acceder a ella, hay que realizar una serie de pasos previos:

En primer lugar, el programa debe habilitar la comunicación con la Kinect. El SDK permite acceder a una lista de todas las Kinect conectadas al ordenador; presuponiendo que para esta aplicación sólo es necesaria una, se elige la primera (y normalmente única) de la lista. Una vez detectada e identificada, es posible iniciar la transferencia de datos desde el sensor hasta la aplicación, pero primero hay que especificar qué datos se quieren recibir.

Los datos principales de la Kinect se transmiten por USB a través de flujos o *streams*. Éstos pueden contener imágenes RGB, imágenes infrarrojas, audio, mapas de profundidad, etc. Cuando un *stream* está activo, lanza eventos periódicos indicando la disponibilidad de un nuevo paquete de sus datos. Para obtener estos datos deben codificarse funciones manejadoras de eventos que capturen estos paquetes, y les den el tratamiento que deseado. Para esta aplicación son necesarios tres *streams*: color, profundidad y esqueletos.

El *stream* de **color** entrega cada fotograma RGB captado por la cámara de la Kinect. No es necesario activarlo para que la detección de esqueletos funcione, pero para el usuario es

un gran apoyo poder ver lo que la Kinect tiene en su campo visual. Para este fin, se ha incorporado a la interfaz gráfica una **ventana** en la que poder mostrar estas imágenes en tiempo real. Se ofrece al usuario la posibilidad de desactivarla, pero no influye de forma perceptible en el rendimiento de la aplicación. También es seleccionable la resolución en píxeles y la frecuencia de captación, pero se recomienda mantener la opción por defecto de 640x480 píxeles y 30 fps.

El *stream* de **profundidad** contiene una imagen análoga a la del sensor RGB, con la diferencia de que cada píxel está caracterizado por un único número y no tres, y que en vez de la luminosidad, su valor indica la distancia en milímetros entre el plano de la cámara y el objeto representado en ese píxel. La interfaz gráfica no da ningún uso a estos datos, pero es necesario mantenerlos en memoria para poder obtener información adicional sobre el usuario, como se detalla más adelante.

El *stream* de **esqueletos** es el verdaderamente importante para la aplicación. En cada evento entrega una lista con todos los usuarios que la Kinect detecte en la imagen. Pueden ser hasta seis, pero sólo dos a la vez pueden estar siendo detectados con todo su esqueleto (en los demás casos, solo detectará la ubicación global del jugador). La aplicación reserva dos variables para guardar esqueletos en su memoria, que la Kinect identifica como pertenecientes al jugador 1 y al jugador 2.

Un paquete de datos sobre esqueletos contiene muchas otras clases de información adicional. Uno de los campos más útiles es la localización del plano del suelo, de la cual se puede extraer la **altura** a la que está colocada la Kinect. Esto, junto con la **inclinación** del sensor (indicada por un acelerómetro) puede ser útil para Blender a la hora de realizar correcciones de posición.

Pero la información realmente valiosa sobre el esqueleto, como se ha apuntado previamente, es la relativa a sus **articulaciones**. Como se ve en la Figura 4, hay veinte de ellas, y cada dos están unidas por un **hueso**. Se considera, además, que existe una jerarquía entre las articulaciones, donde HipCenter es la articulación central, que marca la posición y orientación global del jugador, y cada una es "hija" de la anterior, según el orden de prioridad que indican las flechas azules.

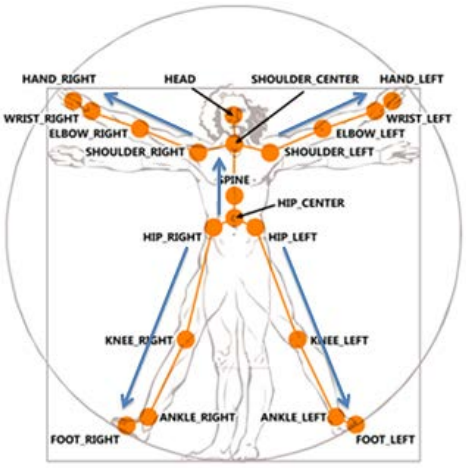

Figura 4 – Articulaciones que forman el Esqueleto Kinect [27]

El dato que caracteriza la posición de las articulaciones es su localización espacial. Ésta se puede expresar de dos formas:

1. Mediante las coordenadas **XYZ** en **metros**, según los ejes mostrados en la Figura 5. En este sistema de coordenadas, la Kinect se encuentra en el origen (0, 0, 0), y el eje Z es paralelo a la dirección en la que apunta el sensor.
2. Mediante la posición del punto que representa una articulación en la imagen RGB, de forma que X e Y son las coordenadas horizontales y verticales en **píxeles**, mientras que Z sigue estando expresada en metros. El punto (X=0, Y=0) marcaría entonces el píxel de la esquina superior izquierda de la imagen. De nuevo, el plano de la Kinect supone el origen de la coordenada Z.

El *middleware* contempla ambas opciones de enviar los datos a Blender, pero la opción más lógica y la usada durante todo este proyecto es la posición espacial real, en metros. Blender, después de todo, utiliza también un sistema de ejes cartesianos XYZ, por lo que la interpretación de las coordenadas es inmediata.

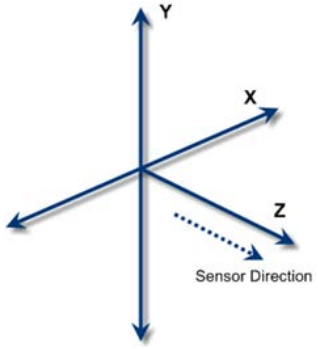

Figura 5 - Colocación de los ejes cartesianos en el espacio de la Kinect

Expresar las coordenadas en píxeles, aun así, resulta útil para la interfaz gráfica, porque facilita la realización de una función importante: **dibujar los esqueletos** sobre la señal de vídeo RGB. Esto permite al usuario comprobar visualmente si está siendo detectado de forma correcta. Al igual que el dibujado de la señal RGB, esta función también se puede desactivar en la interfaz.

Si las articulaciones se caracterizan por su posición espacial, los huesos se caracterizan por su orientación. Las orientaciones, tal como las provee el SDK, se pueden expresar según una matriz de rotación o según un **cuaternión**. Un cuaternión es un número complejo de cuatro dimensiones, que puede utilizarse para representar la dirección de un vector incluyendo la rotación sobre su propio eje. [28] El middleware opta por enviar los cuaterniones siempre, ya que esta forma es una de las opciones contempladas por Blender para expresar rotaciones, y las matrices no. Además de la forma de expresarla numéricamente, también hay dos valores distintos que representan la rotación de los huesos: una es su rotación global, respecto a la cámara. La otra es la rotación jerárquica, es decir, la diferencia entre la rotación de un hueso y la del hueso del que deriva: por ejemplo, la rotación jerárquica del cuello sería lo rotado que esté respecto a la columna vertebral. Se ha preferido utilizar las rotaciones jerárquicas, ya que la herencia entre huesos, es un concepto muy

importante en Blender. Conociendo la orientación global del usuario respecto al sensor, y las orientaciones jerárquicas de cada hueso en la cadena, se podrá deducir exactamente la orientación global de cada hueso.

Otra opción que ofrece la SDK de Kinect es la de realizar la detección de esqueletos en "**modo sentado**", es decir, obtener solo la posición de los brazos y la cabeza. Esta opción mejora la precisión de la detección cuando el usuario está muy cerca de la cámara pero, según la documentación, consume más recursos computacionales. Y, de todas formas, incluso para usuarios en silla de ruedas resulta oportuno evaluar su pose completa, ya que el movimiento del torso jugará un papel importante en varios tipos de ejercicios, y el movimiento de los pies, por leve que sea, podrá ser interpretado por el juego de forma acorde a las capacidades del jugador.

Una herramienta que no forma parte de la SDK básica es la biblioteca KinectInteraction, orientada al manejo de interfaces gráficas mediante gestos de la mano. Sus posibilidades son muchas y ofrece varias herramientas de desarrollo, pero para esta aplicación solo resulta de utilidad una característica concreta: detectar si el usuario tiene las manos abiertas o cerradas en un puño. Su funcionamiento es igual al de los otros *streams*, con una diferencia: no entrega sus datos directamente al manejador, sino que para obtener el estado de las manos, debe realizar un procesado conjunto de los *streams* de esqueleto y de profundidad.

3.1.2.2. La comunicación con Blender

Una vez guardados en memoria los esqueletos y toda la información adicional sobre el estado de la Kinect, es necesario transmitir los datos a Blender. El sistema de comunicación se basa en el usado por la aplicación KinectOSC, que ha servido de modelo para este proyecto. Originalmente se codificó el receptor en Blender para recibir los mensajes enviados por KinectOSC, y después se creó el middleware Chiro para suplir mejor las demandas de este proyecto. A partir de ese punto se fueron mejorando emisor y receptor conjuntamente.

Para enviar la posición del usuario al juego hay que codificar los datos en un paquete y enviarlo a un puerto de la dirección IP local. Para enviarlos se ha elegido el protocolo UDP (*User Datagram Protocol)*, puesto que para implementar un control preciso es vital que haya un retardo mínimo, mientras que la pérdida de información puntual no supone un problema importante.

Si UDP es el método de envío, la información está codificada según el protocolo OSC (*Open Sound Control*). OSC es "un protocolo abierto, independiente del transporte, basado en mensajes, desarrollado para la comunicación entre ordenadores, sintetizadores de sonido y otros dispositivos multimedia" [29]. Se trata, en esencia, de un equivalente al protocolo MIDI. Permite enviar números enteros, flotantes, o cadenas de texto ASCII, con la condición de que todos los campos deben tener una longitud múltiplo de 4 bytes. Si no llegan a este número, OSC añadirá caracteres nulos al final hasta que se cumpla.

Los paquetes OSC pueden adoptar dos formas, *message* o *bundle*. La elección del formato *bundle*, al igual que la del propio protocolo OSC para transportar los datos, se ha hecho por herencia de la aplicación KinectOSC.

Un *bundle* tiene una longitud variable, pero una serie de campos fijos, representados en la Figura 6:

- Una cadena ASCII de 8 bytes, con el texto "#bundle".
- Una etiqueta temporal de 8 bytes.
- Uno o más *OSC Bundle Elements*. Éstos no son otra cosa que un número entero seguido de más *messages* o *bundles*, introducidos recursivamente en el *bundle* principal. El entero indica la longitud en bytes del *message* o del *bundle*. En este caso sólo es necesario introducir un *message*, que se compone a su vez de los siguientes campos:
    - Un **patrón de dirección** OSC, que no es otra cosa que un texto ASCII que empiece por el carácter "/". En esta aplicación se ha aprovechado este campo para indicar la articulación, hueso o manos cuyos datos se están enviando. Cada *bundle* contendrá información sobre uno sólo de estos elementos.
    - Una **etiqueta de tipo**. Se trata de un texto que indica el número y tipo de variables que vienen a continuación: una "s" indica una cadena de texto, una "i" un número entero y una "f" un número flotante. Por ejemplo, la etiqueta ",fff" indicaría que a continuación hay tres números de punto flotante, mientras que ",iis" indicaría que hay dos números enteros y una cadena de texto.
    - Las variables especificadas por el campo anterior.

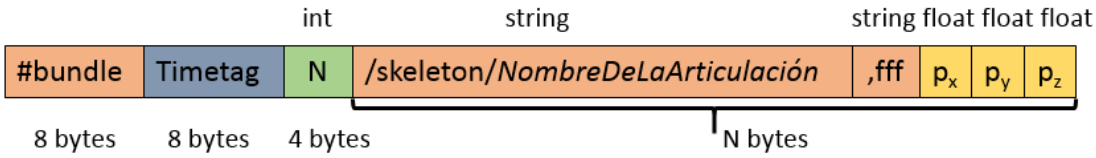

Figura 6 - Estructura de un bundle *OSC*

Para esta aplicación se ha optado por organizar las variables en función de la información que se quieran expresar, de forma que al leer la etiqueta de tipo el receptor sepa unívocamente qué datos está recibiendo. Las modalidades incluidas son:

- ",fff". Tres números flotantes. Son las coordenadas espaciales, expresadas en metros, de una articulación: X, Y y Z, necesariamente en este orden. El patrón de dirección de este mensaje será */skeleton/NombreDeLaArticulación*.
- ",fffff". Cinco números flotantes. Son las coordenadas espaciales de una articulación, con X e Y expresadas en píxeles, tal como se ha explicado en el punto 3.1.2.1, y con Z expresada en metros. Para diferenciar este mensaje de uno que contenga las tres coordenadas en metros, se han añadido dos números flotantes de valor nulo al final.
- ",ffff". Cuatro números flotantes. Son las cuatro componentes de un cuaternión, que expresa la rotación jerárquica de un hueso. El patrón de dirección tendrá el formato */skeleton/ArticulaciónFinal*, donde *ArticulaciónFinal* representa la articulación que

marca el final del hueso. Por ejemplo, la rotación del cuello estaría precedida por la cadena */skeleton/Head*, y la del antebrazo derecho por */skeleton/WristRight*. Indicar la articulación final en lugar de la de origen es poco intuitivo, pero evita las confusiones a la hora de enviar una rotación con la cadena */skeleton/HipCenter*: es la orientación global del jugador, no la del hueso que une las articulaciones HipCenter y Spine.

- ",ii". Dos números enteros. Son los estados de las manos del jugador, abierta (0) o cerrada en un puño (1). Pese a tratarse de una variable binaria, se expresa como un valor entero: OSC no permite enviar booleanos, ya que todos sus campos tienen una longitud mínima de 4 bytes. El primer número en ser enviado representa la mano izquierda, y el segundo la derecha. El patrón de dirección será invariablemente */skeleton/Fists* o */skeleton/FistsOld*. La diferencia entre ambos se aclara más adelante.

Como ya se ha visto, la información sobre el usuario se actualiza cada vez que la Kinect lanza un evento avisando de la disponibilidad de nuevos datos. Esto ocurre a una frecuencia estándar de 30 fps, que afortunadamente es múltiplo de la frecuencia de ejecución de los juegos en Blender, 60 fps. Sin embargo, esta última es solo una frecuencia nominal y puede variar mucho en función de la carga gráfica del juego y la potencia del ordenador. Si el *middleware* enviara mensajes sólo cuando los datos del usuario se actualizan, ambos sistemas correrían el peligro de desincronizarse, y el rendimiento del juego se vería reducido.

Por eso da mejores resultados dejar que sea Blender quien gobierne la comunicación. En uno de cada dos fotogramas enviará un mensaje de petición al middleware, y éste responderá con los datos más recientes que tenga sobre la posición del usuario. Cada juego de Blender puede precisar una clase distinta de información, por lo que el *middleware* enviará, ante cada petición, las posiciones de las articulaciones, las rotaciones de los huesos, o ambos conjuntos. Adicional e invariablemente, enviará también los estados de los puños del jugador.

Este sistema no sólo evita que el juego se ralentice, sino también que se bloquee si por cualquier causa la Kinect deja de detectar al usuario, por ejemplo, si alguien pasa por delante y tapa la cámara. Si esto ocurre, el *middleware* dejará de recibir actualizaciones, pero al menos al recibir peticiones de Blender tendrá datos que enviar: los últimos almacenados. Por otra parte, también es útil para Blender saber si está recibiendo datos repetidos: por eso, si el *middleware* envía el mismo esqueleto con la misma posición dos o más veces, lo indicará cambiando el patrón de dirección de los puños de *Fists* a *FistsOld*. Blender puede comprobar esta cadena de texto y actuar como el desarrollador del juego estime oportuno, por ejemplo mostrando una pantalla de pausa.

Como se ha dicho anteriormente, la comunicación entre Blender y el *middleware* permite incluso realizar la configuración de este último de forma automática. Blender enviará un mensaje con una serie de parámetros de configuración, que dependerán del entorno del juego, y el *middleware* responderá, después de hacer los ajustes oportunos, con otros datos adicionales de interés para el juego. Los parámetros dictados por Blender son:

- El tipo de información que necesita el juego: posiciones, rotaciones o ambas.
- La necesidad de datos de un solo jugador o de dos.
- Player1Port. El puerto al que el *middleware* debe enviar los datos del esqueleto del jugador 1.
- Player2Port. El puerto al que enviar los datos del jugador 2, si procediera.
- BlenderPort. El puerto al que enviar la respuesta a este mensaje de configuración concreto.

A su vez, los parámetros con los que responde el *middleware* son solo dos: la altura del sensor respecto al suelo, en metros, y su inclinación en grados.

Antes de recibir un mensaje de configuración, Blender abre por defecto el socket UDP de envío al jugador 1 en el puerto 8001 y el del jugador 2 en el 8002. Los mensajes de configuración, al igual que los de petición de datos, son recibidos mediante un socket UDP asíncrono que está en constante escucha, en el puerto MiddlePort. Cuando recibe un paquete, se activa una función que detecta si lo que ha recibido es una petición de datos o una orden de configuración, y actúa en consecuencia. Si hay que hacer una reconfiguración, se cierran y destruyen todos los sockets abiertos y se crean otros nuevos, por los que se realizará el envío y la recepción a partir de ahora.

Hay, por lo tanto, cuatro puertos en juego: los de envío a los jugadores 1 y 2, el de recepción de las peticiones de datos y las órdenes de configuración, y el de envío de la respuesta a la configuración. Todos menos este último son configurables manualmente por el usuario mediante la interfaz gráfica. Al fin y al cabo, el envío de la respuesta siempre va a ir precedida por la recepción de una orden de configuración, donde se especifica el puerto al que debe enviarse. Por su parte, el puerto de escucha no es configurable por Blender, ya que para hacerlo primero tendría que enviar un mensaje a ese mismo socket. Una paradoja que sólo puede resolverse si el número de este puerto es una constante invariable durante toda la sesión de juego (por defecto, el 8000).

La Figura 7 ilustra mejor el protocolo de comunicación que se acaba de describir. En la primera fase (arriba) tiene lugar la recolección de datos de esqueleto por parte del middleware. En la fase 2 (en medio) se realiza la configuración automática, y la fase 3 (abajo) contiene el envío regular de datos.

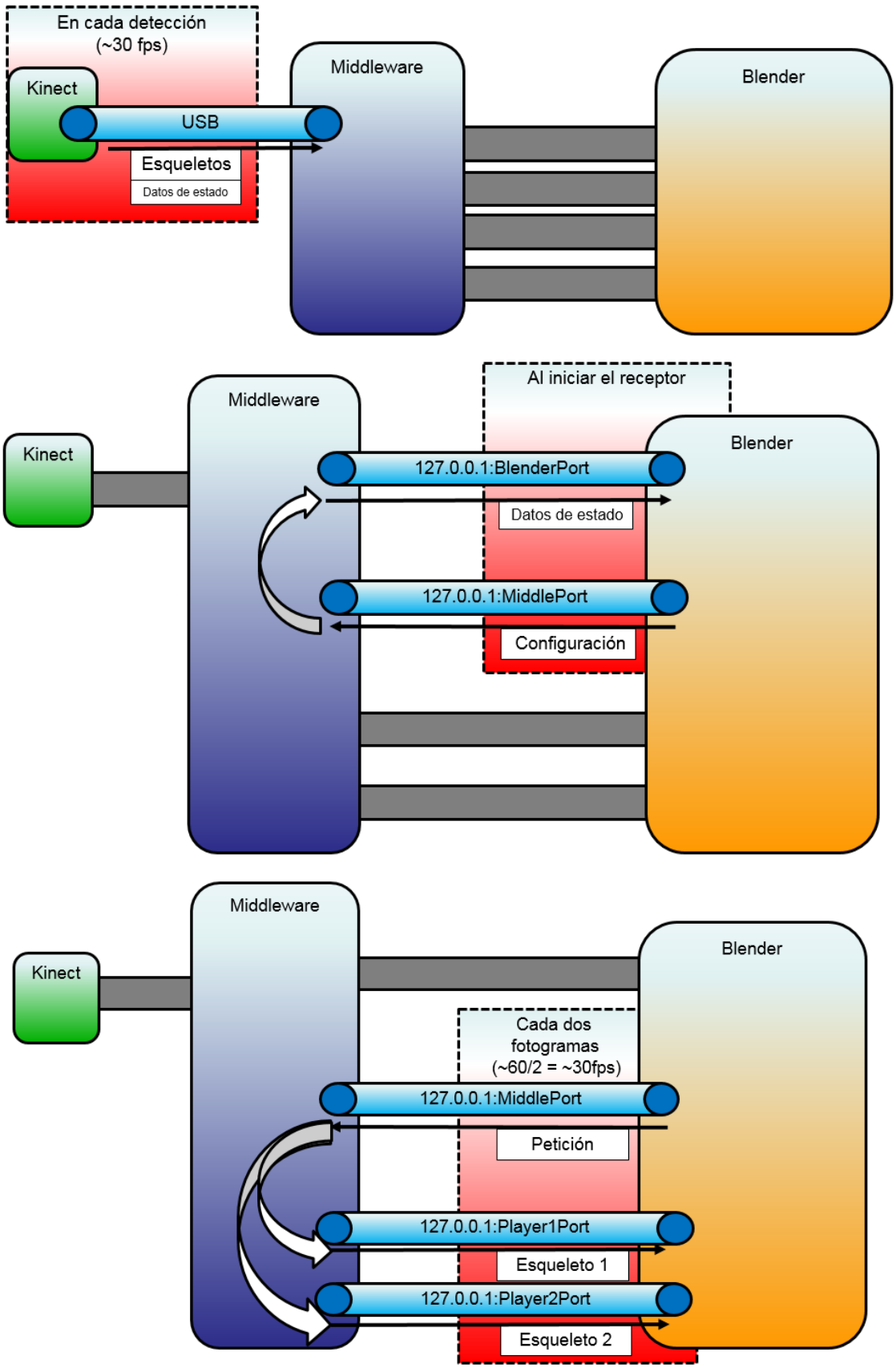

Figura 7 – Diagrama de bloques de la comunicación entre el middleware KinectWindow y Blender. Arriba: recolección de datos de esqueleto. Medio: configuración automática. Abajo: envío regular de datos.

Las respuestas del middleware a los mensajes de configuración se codifican también mediante el protocolo OSC, siendo su patrón de dirección */skeleton/Environment* y las variables enviadas ",fi": un número flotante, la **altura** del Kinect, y uno entero, su **inclinación**.

Los mensajes enviados por el juego al middleware, en cambio, se componen de cadenas de bytes, que el *middleware* debe descomponer en números enteros o secuencias de bits. La Figura 8 ilustra sus posibles estructuras, que se componen de los siguientes campos de datos:

- La **cabecera**. Se compone de cuatro bytes que el middleware debe interpretar como un número entero, de valor únicamente simbólico y muy visual para poder distinguir unos mensajes de otros. Si el número es 876543210, se trata de una petición de datos de usuario, y no lo seguirán más campos. Si la cabecera es 1234567890, es una orden de configuración, y lo seguirán los siguientes datos:
- Options. Éste es un solo byte, con cinco bits en blanco y tres significativos. Su posición invariable permite leerlos de forma unívoca. El bit que ocupa la tercera posición indica si se necesitan datos de un **segundo jugador** o no. Los dos primeros bits indican el **tipo de datos** que el juego necesita en ese momento:
    - 00 ó 11: Datos de **rotación** de todos los huesos, y **localización** sólo de la articulación HipCenter.
    - 01: datos de **localización** de todas las articulaciones.
    - 10: datos de **rotación** de todos los huesos, y **localización** de todas las articulaciones.
- BlenderPort. Cuatro bytes que forman un valor entero, el del puerto al que se debe enviar la respuesta a este mensaje.
- Port1 y Port2. Otros dos enteros, los números de los puertos a los que enviar los datos de los esqueletos.

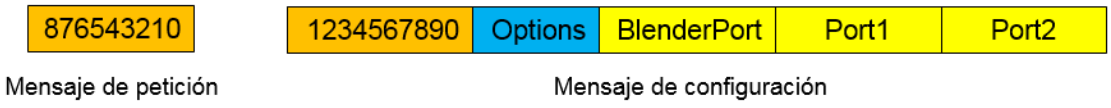

Figura 8 - Mensajes enviados por Blender al middleware

3.1.3. Otros middlewares implementados

Al término de este proyecto, KinectWindow es sólo una de las tres pestañas disponibles para el Chiro. De las otras dos, la más completa es el módulo HMD (*Head Mounted Display)*, desarrollado por José Zarco en su Proyecto de Fin de Grado [25].

El HMD funciona de forma muy similar a la KinectWindow: se apoya en un entorno de desarrollo preexistente, el OSVR (*Open Source Virtual Reality*) [30] para obtener los datos entregados por el **Oculus Rift** [9], que en su caso es únicamente un cuaternión expresando la orientación de la **cabeza** del usuario. Este dato debe ser transmitido a Blender para representar la perspectiva del jugador dentro del entorno 3D. Para ello utiliza el mismo

sistema que la KinectWindow: un socket UDP en el *middleware* espera la llegada de peticiones de datos, y responde a ellas enviando el cuaternión por otro socket, éste de envío. OSVR gestiona el manejo del casco de forma autónoma, por lo que no necesita control por parte del usuario. Esto implica que tampoco es necesaria hacer la fase de configuración que sí realiza el módulo de Kinect. Como se observa en la Figura 9, la interfaz del HMD sólo necesita que el usuario introduzca el puerto de escucha y el puerto de envío. En esta figura, además, puede apreciarse en la parte superior del Chiro el menú que permite mostrar la pestaña HMD o la KinectWindow.

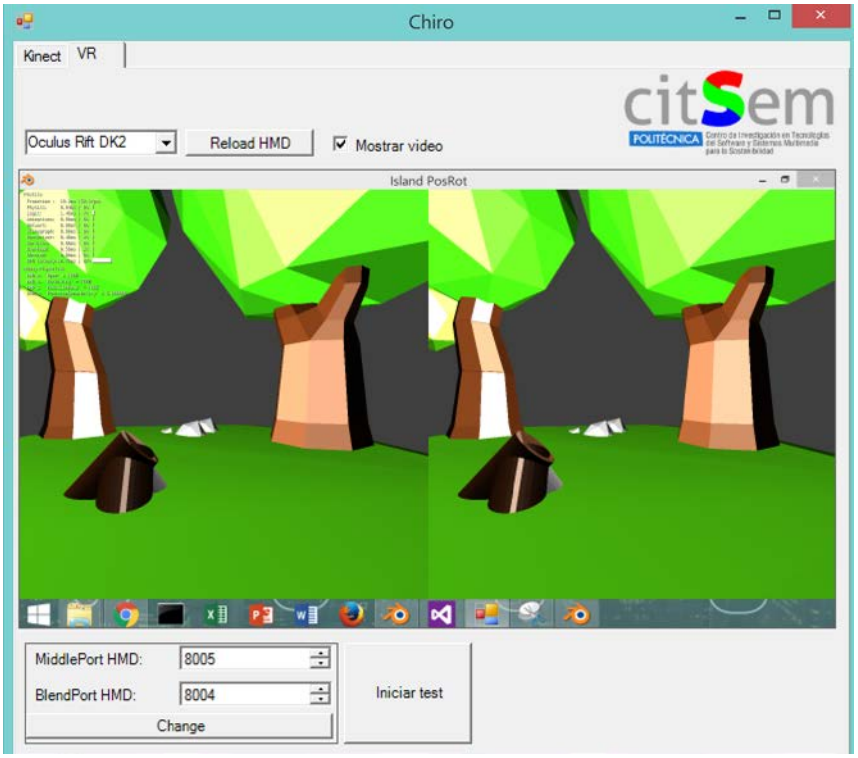

Figura 9 - Interfaz de la pestaña HMD

El otro *middleware* disponible sí ha formado parte de este PFG, y se le ha dado el nombre de PhoneWindow. Su objetivo es permitir el manejo en Blender de un objeto que represente un dispositivo móvil, y refleje su movimiento en manos del usuario. Ésta es otra clase de control por movimientos, popularizada por la consola Wii, que no refleja el movimiento del cuerpo del jugador, sino el de un objeto en sus manos.

Para ello establece un canal de comunicación con dispositivos móviles con sistema operativo Android, obtiene sus datos de orientación y se los transmite al juego. Al igual que con la KinectWindow, se han desarrollado dos facetas de su funcionamiento: la conexión con el móvil y la conexión con Blender.

La conexión con el dispositivo móvil se hace a través de un cable USB. Existe una aplicación, el ADB (*Android Debug Bridge*) [31] que permite la comunicación entre puertos de igual número del *smartphone* y del ordenador: para este uso se ha elegido el puerto 9000.

A diferencia de la Kinect y del Oculus Rift, que actúan como interfaces a las que puede acceder directamente el *middleware*, en un dispositivo Android se debe desarrollar una

aplicación que acceda a los datos que sean necesarios y los envíe al middleware del PC. A esta aplicación, cuya interfaz se puede ver en la Figura 10, se le ha dado el nombre de ADBRotation, y se ha desarrollado en lenguaje Java en el entorno de desarrollo de Android [32], utilizando el programa AndroidStudio para Windows. ADBRotation crea un servidor TCP (*Transmission Control Protocol*) que espera una solicitud de conexión del puerto 9000. Cuando la recibe, empieza a mandar periódicamente los datos de la rotación del móvil.

Estos datos se obtienen combinando, mediante una función disponible en la API (*Application Programming Interface*) de Android, los datos entregados por dos sensores: el del campo magnético terrestre, normalmente utilizado para las aplicaciones de brújula, y el de gravedad, que es un uso específico del acelerómetro. La gran mayoría de dispositivos Android incorporan estos dos sensores. Combinando estos dos datos se puede obtener el cuaternión que expresa la rotación del móvil, tomando como referencia un vector paralelo al suelo que apunte al polo norte magnético.

Cada vez que estos datos se actualizan, son enviados por TCP al cliente que los está esperando en el *middleware* del ordenador. Éste tiene dos hilos paralelos en ejecución: uno dedicado a la recepción de estos datos y su almacenamiento en la memoria temporal, y otro en constante comunicación con Blender. Al igual que la KinectWindow y el HMD, abre un socket de escucha UDP que recibe constantemente peticiones de datos por parte de Blender, y por otro socket distinto responde enviando los cuaterniones.

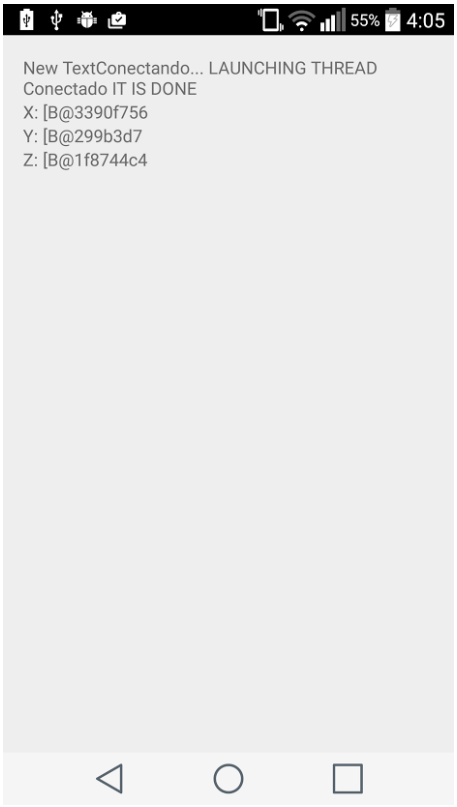

Figura 10 - Interfaz en desarrollo de la app ADBRotation

3.2. El receptor de Blender

Si el *middleware* se ocupa de leer la posición del usuario y transmitirla, es en Blender donde se debe implementar el sistema de control para videojuegos que utilice el movimiento del jugador. Se trata de un programa de edición en 3D, que permite modelar objetos y animarlos con una calidad casi fotorrealista, y exportar o bien archivos de vídeo o bien aplicaciones interactivas ejecutables, que es lo que interesa en este caso.

El desarrollo de juegos se realiza mediante el Blender Game Engine (BGE) [8], una herramienta que permite crear escenas jugables combinando objetos 3D a los que asigna modelos físicos, animaciones, iluminación y, lo más importante, lógica de comportamiento. Ésta se compone de una serie de propiedades asignables al objeto (numéricas, de texto o booleanas) y de su combinación con acciones y condiciones. Se puede configurar, por ejemplo, el movimiento de un personaje, asignarle un contador de puntos de vida, hacer que este contador baje al entrar en contacto con otro personaje, y eliminarlo por completo de la escena cuando llegue a cero.

Dentro de las acciones realizables por un objeto está la posibilidad de ejecutar scripts codificados en lenguaje Python. Esto es especialmente útil para realizar operaciones demasiado complejas para programarlas en el editor gráfico del BGE.

Los scripts de Python no sólo pueden ser llamados desde un juego en ejecución, sino que pueden ejecutarse desde el propio editor del BGE para incorporar nuevas funcionalidades a Blender, e incluso modificar su interfaz gráfica para añadir botones personalizados. Estos archivos de código se conocen como ***add-ons***, y pueden instalarse en el propio Blender para incorporar sus añadidos permanentemente, tal y como se detalla en el Anexo 1.

Y esto es lo que se ha desarrollado: un *add-on* que contiene, por una parte, todas las funciones necesarias para implementar un control por movimientos desde el editor, y por otra, los scripts a los que el juego necesitará acceder durante su ejecución para recibir datos constantemente.

En la Figura 11 se puede ver la nueva interfaz creada en Blender al instalar el *add-on*. Los cuatro primeros botones sirven para añadir al entorno 3D distintas variantes del objeto receptor, el **Esqueleto Kinect**, cuyo funcionamiento se detallará enseguida. Los otros dos, la cuerda y el acelerómetro, son **objetos auxiliares** para integrar los movimientos del esqueleto en el esquema de control de Blender. Se explicará su función exacta más adelante.

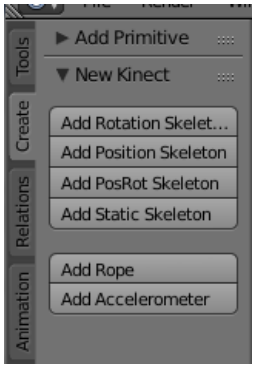

Figura 11 - Interfaz de Kinect creada por el add-on

3.2.1. Asignación del movimiento del jugador a un esqueleto

Toda la implementación del receptor gira alrededor de un elemento principal: la **armadura**. Una armadura es una clase de objeto de Blender que representa la estructura básica de un cuerpo, humano o de cualquier otra clase. Está compuesta de "**huesos**", cuya forma es invariablemente la de una línea recta, en la que sin embargo hay marcada una cara delantera. Su uso habitual es simplificar la animación de cuerpos, asociando distintas partes de una malla 3D a cada hueso y luego animando simplemente el movimiento de éste, en lugar de cada vértice individual de la malla. Los huesos suelen derivar unos de otros en una relación que Blender llama de "padres" e "hijos", por lo que es posible mover una extremidad entera con varios huesos animando únicamente aquél del que derivan todos los demás.

Otra forma de controlar el movimiento de los huesos es mediante el uso de unas órdenes llamadas *constraints*, **restricciones**, que pueden obligar a un hueso a apuntar siempre en una dirección determinada o hacia un objetivo concreto, o imitar la rotación de otro hueso u objeto. Estas restricciones serán importantes a la hora de implementar el sistema de control, ya que muchas veces es deseado que un hueso del personaje principal sea gobernado por el movimiento del esqueleto receptor.

La función primaria del *add-on*, entonces, es crear una armadura que reciba e interprete los datos que envía el *middleware*. Para eso, por supuesto, la forma de la armadura debe coincidir con la del esqueleto básico detectado por Kinect. Puesto que Kinect detecta hasta dos usuarios, en una misma escena del juego se pueden añadir hasta dos receptores. En la Figura 12 se puede ver el aspecto del objeto resultante, y en la Figura 13 se aprecia el cambio que experimenta el panel de controles de Kinect: ya sólo se puede añadir otro receptor, que debe ser del mismo tipo que el ya existente. Aparece la opción de guardar una animación, de añadir un amplificador a una articulación determinada, y los controles manuales de las propiedades MiddlePort y BlenderPort que se mencionaron en el apartado 3.1.2.2.

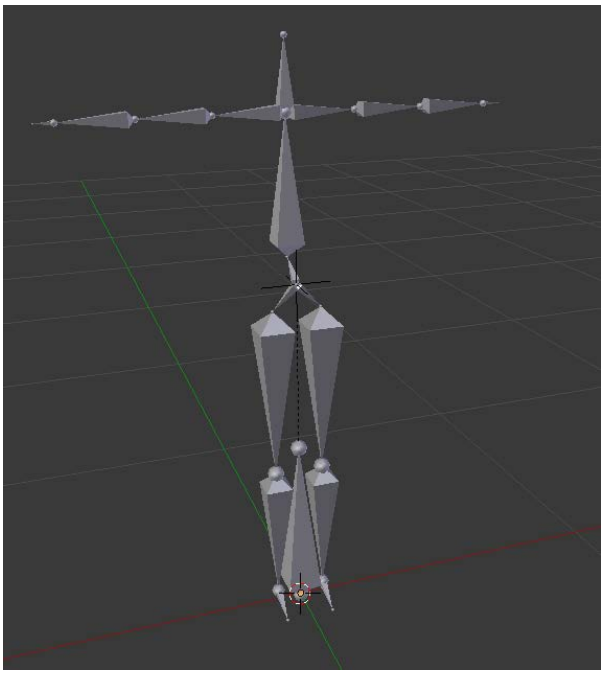

Figura 12 - Esqueleto Kinect de tipo Rotación

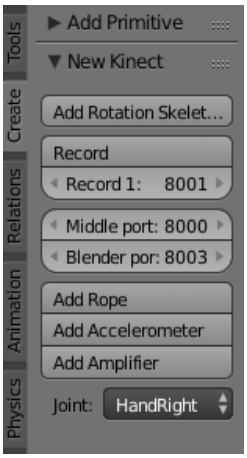

Figura 13 - Interfaz de Kinect con un receptor ya añadido a la escena

Al pulsar uno de los cuatro botones principales de la interfaz original, se llama a una función que introduce en el entorno 3D de Blender una armadura básica con un solo hueso, y va añadiendo huesos y controlando su longitud, sus relaciones y su pose hasta completar lo que será el esqueleto del usuario.

El esqueleto que se ve en la Figura 12 es el primero de los cuatro tipos de esqueleto receptor, el Esqueleto de Rotación. Los otros dos son el de Posición, el Combinado, y el Estático, que se utiliza para la amplificación de movimientos. Más adelante se detallarán las diferencias entre ellos.

En la Figura 14 se puede ver la lógica de juego estándar que el *add-on* crea asociada al esqueleto receptor. Existen una serie de propiedades, y dos sensores que activan diferentes acciones:

- Al iniciarse la escena en la que se encuentra el receptor:
    - Se activa el movimiento en la armadura. Éste es un paso necesario en todas las armaduras de Blender.
    - Se llama a la función de apertura del socket. Esta función utiliza la librería de sockets de Python para abrir un socket UDP de recepción en el puerto indicado por la propiedad **Port**. Si la apertura tiene éxito, la propiedad **Open** cambiará a valor *True*, verdadero.
    - Si éste es el primer receptor que se ha añadido a la escena, se llama a la función de configuración. Esta función abre un socket de envío en el puerto MiddlePort, y envía un mensaje de orden de configuración según el formato explicado en el punto 3.1.2.2. Para ello, comprueba la propiedad **Player** de todos los receptores presentes, que los identifica como pertenecientes al jugador 1 o al jugador 2, la propiedad **Type**, que indica el tipo de datos que debe recibir y debe ser igual en ambos, la propiedad Port de ambos, y la propiedad BlenderPort tal como se encuentra definida en la interfaz. En el puerto que esta última indica, abrirá un socket de recepción para obtener la altura y la inclinación del sensor Kinect. Utilizando estos datos, ajustará la

colocación del esqueleto para que el plano del suelo real coincida con el del entorno 3D.

- En uno de cada dos fotogramas, si la propiedad Open está activa, es decir, si el socket de recepción de este esqueleto está abierto, se activa la función de recepción. Ésta hará un intento de recibir datos por ese socket, y si lo consigue antes de un tiempo determinado (500 ms), la propiedad **Receiving** se reescribirá con el estado *True*, y si no pasará a *False*. utilizará una clase obtenida de Internet para leer los datos tal y cómo están codificados según el protocolo OSC. Según la etiqueta de tipo sabrá si está recibiendo coordenadas, cuaterniones o estados, y según el patrón de dirección sabrá a qué hueso o articulación pertenece. Si se trata del estado de los puños, actualizará las propiedades **LeftFist** y **RightFist**, y leerá el patrón de dirección para saber si se están recibiendo datos desactualizados. Si es el caso, la propiedad **Repeat** aumentará su valor en 1, y si no se reiniciará a cero. El propósito de esto es que el desarrollador pueda, si lo desea, tomar medidas si el valor de Repeat sobrepasa cierto límite.

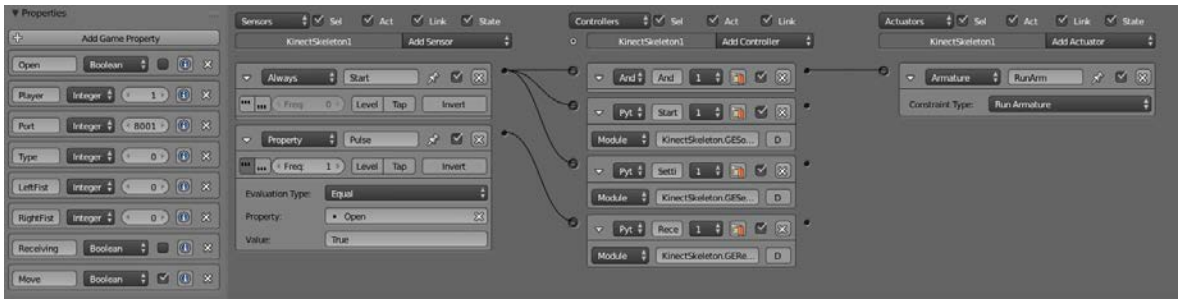

Figura 14 - Lógica asociada al receptor

3.2.2. Clases de esqueleto receptor

Los tipos de esqueleto indican el tipo de datos de pose que reciben, y están construidos de forma acorde. El **Esqueleto de Rotación** (tipo 0), por ejemplo, solo recibe los cuaterniones de los huesos, más los estados de los puños (eso es constante a todos los esqueletos) y la posición en XYZ de la articulación HipCenter solamente. Esto permite al esqueleto imitar el movimiento espacial del jugador, pero es posible desactivar esta opción y mantenerlo quieto mediante la propiedad **Move**. Para todos los tipos de esqueleto, si Move está a *False*, HipCenter se mantendrá fija. El esqueleto de rotación incorpora un hueso adicional, HipCenterRot, que representa la orientación global del usuario. De este hueso dependen todos los demás, y si se rota éste los demás rotarán con él. Todos los huesos están conectados y heredan la rotación de sus padres, por lo que al recibir la rotación jerárquica, la pose se corresponde exactamente con la del usuario. El inconveniente del esqueleto de rotación es que la longitud de los huesos es fija y constante para todos los esqueletos de este tipo, por lo que rara vez reflejará la anatomía real del usuario ni la amplitud real de sus movimientos. Esto no suele ser un problema, ya que este esqueleto está pensado para que un esqueleto de personaje copie sus rotaciones, y el personaje también va a tener sus propias proporciones.

Para los casos donde la magnitud del movimiento real sí importe, está el **Esqueleto de Posición** (tipo 1), formado por veinte objetos *Empty* (que representan puntos en el espacio) que reciben la posición de las articulaciones. Las posiciones se expresan siempre respecto a la articulación de la que deriva cada una, por lo que si se desactiva el movimiento de HipCenter, la relación entre las posiciones seguirá siendo correcta. Este esqueleto sigue teniendo huesos, por supuesto, pero están desconectados entre sí y cada uno tiene dos restricciones: una le hace tomar la posición de su articulación de origen, y la otra le hace apuntar hacia la siguiente articulación de la jerarquía. El problema de este sistema es que asigna un valor relativamente aleatorio a la rotación de los huesos sobre sí mismos y no tiene en cuenta la orientación global del jugador, por lo que a la hora de trasladar estas rotaciones a un personaje, el efecto visual puede ser aberrante. Por ejemplo, puede ser que el personaje tenga un hombro proyectado hacia delante y otro hacia atrás, pero que el torso siga mirando hacia delante en lugar de girarse como es debido.

El **Esqueleto Combinado** (tipo 2) soluciona los problemas de los dos esqueletos anteriores: en esencia es igual que el de posición, pero sustituye la restricción de apuntado de los huesos por la recepción de los datos de orientación reales, e incorpora de nuevo el hueso que representa la rotación global. De esta forma, tanto las rotaciones como las posiciones de las articulaciones son exactas. Este esqueleto entrega la representación perfecta de los movimientos del jugador, a cambio de ser mucho más complejo y necesitar un flujo de datos de casi el doble de volumen que los anteriores.

El Esqueleto Estático (tipo 3) está orientado a la amplificación de movimientos, por lo que la explicación de su funcionamiento se deja para el apartado correspondiente. Si en la Figura 12 se mostró la forma del esqueleto de rotación, a continuación se presentan el de posición y el combinado (Figura 15).

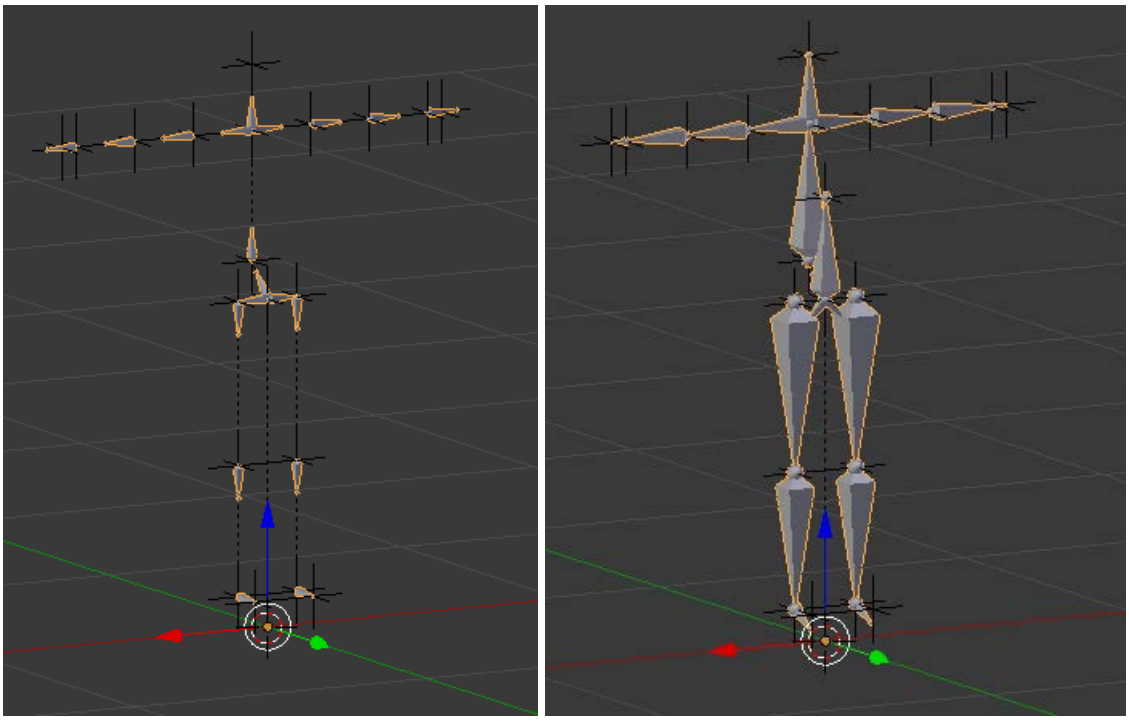

Figura 15- Esqueleto de posición, esqueleto combinado

3.2.3. Animación y grabación de movimiento

Además del control en Blender Game Engine, e imitando a otros softwares de captura de movimiento, se ha implementado la posibilidad de recibir datos de la Kinect en el editor 3D de Blender, de tal forma que se crea una animación que es posible reproducir una y otra vez. Esto puede simplificar enormemente el proceso de animación de los personajes humanos, puesto que el desarrollador sólo tendrá que grabarse con la Kinect realizando con mímica la acción que desee, y luego podrá guardar ese movimiento en la memoria del juego.

Este proceso se activa con el botón "**Record**" de la interfaz del *add-on*, que aparece cuando hay al menos un esqueleto receptor añadido a la escena. Cuando hay dos, una casilla de selección da la opción de realizar la grabación de ambos o de uno solo. Fuera de la ejecución del juego es imposible acceder a las propiedades lógicas de un objeto, por lo que se añaden dos campos adicionales con los que especificar los puertos de recepción de ambos esqueletos. El tipo del esqueleto queda especificado en la lógica interna del *add-on*, y no es necesaria que sea visible al jugador.

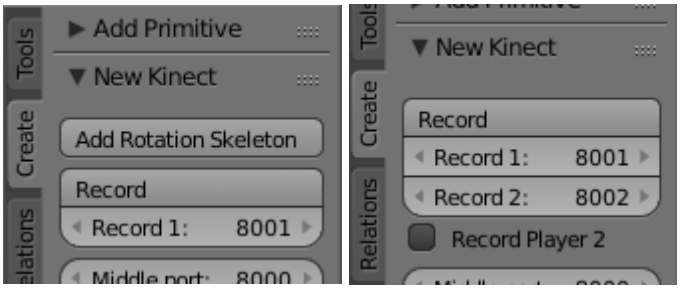

Figura 16 - Opciones de grabación, con uno y con dos esqueletos en la escena

La mayoría de opciones de visualización del editor 3D incluyen en la parte inferior una línea de tiempo que representa la animación actual, tal como aparece en la Figura 17. Esta sección incluye controles para reproducir y pausar la animación, así como especificar sus fotogramas de inicio y fin, es decir, su duración. La duración de los fotogramas se puede variar, pero la opción por defecto es 24 por segundo, como en cine. La barra verde representa el fotograma actual, mientras que una línea amarilla representa un **fotograma clave** de un objeto. Éstos son fotogramas en los que queda fijado un valor para una propiedad determinada del objeto; si en la línea de tiempo se colocan dos fotogramas clave con valores distintos para la misma propiedad, Blender interpolará los valores en cada fotograma de la línea de tiempo para que haya una transición uniforme.

Por ejemplo: suponiendo que hay un cubo seleccionado y que la animación dé comienzo en el fotograma 1 y termine en el 100. Se coloca un fotograma clave para la propiedad de Localización, con el valor XYZ = (0, 0, 0), en el fotograma 10. Se coloca otro fotograma clave en el fotograma 50 con el valor (0, 0, 1), y otro con el valor (0, 0, -0,5) en el fotograma 60. Al reproducir la animación, durante los primeros diez fotogramas el cubo permanecerá fijo en el origen de coordenadas. Al llegar al fotograma número diez, empezará a subir con velocidad constante hasta alcanzar un metro de altura en el fotograma 50, y después bajará más rápidamente hasta llegar 50 cm por debajo de su posición inicial en el fotograma 60, donde permanecerá hasta el final de la animación.

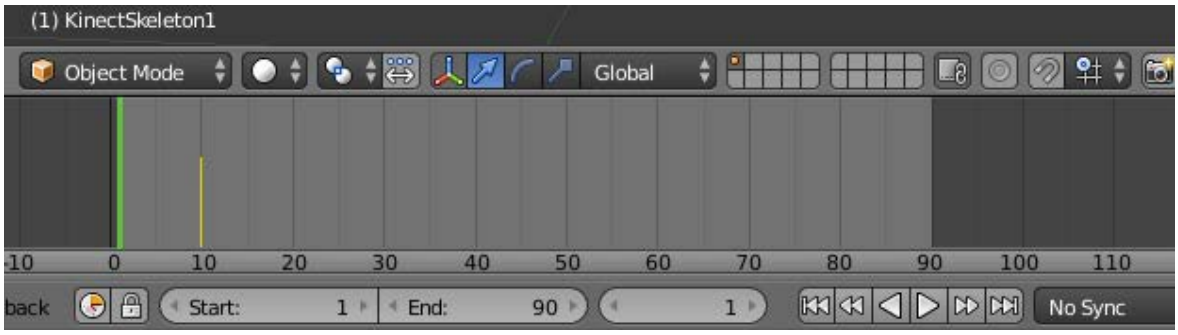

Figura 17 - Línea de tiempo

Cuando se activa el botón de grabación, se llama a un script que, tras enviar la orden de configuración y procesar la respuesta, empieza a avanzar por la línea de tiempo. En cada fotograma recibe datos y crea un fotograma clave con la propiedad recibida: rotación de un hueso, posición de una articulación o ambas (el estado de los puños no se implementa en esta variante, ya que Cuando se alcanza el final predeterminado de la animación, la recepción termina.

No es aconsejable guardar las animaciones asociados al propio esqueleto receptor, puesto que es difícil predecir si entrará en conflicto con la recepción real de los datos. Lo que sí se puede hacer es, nada más realizar la grabación, utilizar restricciones para que un personaje con esqueleto propio copie el movimiento del receptor, y guardar esa animación del personaje. Esto tiene la ventaja adicional de permitir al usuario colocar los fotogramas clave sólo en los puntos que considere necesarios, ya que 24 fotogramas clave por segundo es una exageración sin sentido.

3.2.4. Control de juego

Se ha explicado la forma en la que un juego desarrollado con Blender puede recibir datos y trasladar los movimientos del jugador a un esqueleto receptor. El modelo 3D del personaje principal del juego puede copiar estos movimientos en tiempo real. Pero no basta sólo con copiarlos. El juego debe poder reconocer cuándo se están ejecutando movimientos determinados que activen acciones: por ejemplo, distinguir un simple movimiento con el pie de una patada a un enemigo, cuándo la mano está tocando un objeto con el que debe interactuar, o si el jugador está dando la orden de abrir un menú.

Todo esto es fácil de conseguir utilizando scripts de Python, o creando objetos personalizados para la ocasión con una lógica que reaccione a los movimientos del esqueleto. Se pretende, sin embargo, que esta interfaz sea fácil de utilizar para cualquiera con un mínimo nivel de manejo del Blender Game Engine, sin que los conocimientos de Python sean un requisito. En la mayoría de los casos, de todos modos, las interacciones con el esqueleto se limitarán a medir distancias y velocidades en los movimientos. Por esta razón se ha añadido a la interfaz la creación rápida de dos objetos que cubrirán estas funciones: el acelerómetro y la cuerda.

3.2.4.1. Objetos auxiliares

El **acelerómetro** es sencillamente un cubo pequeño que mide la velocidad a la que se mueve. Para ello se le ha dotado de una propiedad, Work, que al ponerse a *True* inicia la ejecución periódica a 30 fps de un script de Python. Este script accede a una función de la API de Blender para obtener la velocidad a la que se está moviendo el objeto. Nótese que aunque sí se utiliza Python para hacer funcionar este objeto, la función viene ya definida en el *add-on* y el desarrollador no tiene que interactuar con nada más que con los valores que entrega. Estos valores son escritos en las propiedades de juego del acelerómetro, que representan:

- La velocidad global en componentes vectoriales: **Vx**, **Vy** y **Vz**. Es la velocidad a la que se mueve el acelerómetro, expresada según los ejes XYZ del entorno 3D. Calculando el módulo de este vector se obtiene la velocidad global, **V**. Todas las velocidades se expresan en metros por segundo.
- La velocidad local en sus componentes **Vpx**, **Vpy** y **Vpz**. La velocidad tiene la misma magnitud que en la propiedad anterior, pero en lugar del entorno global se expresa según los ejes locales XYZ del acelerómetro: y es que cada objeto de Blender tiene unos ejes XYZ propios, que rotan cuando el objeto lo hace, y la orientación del cubo rara vez será constante en una escena. Por ejemplo, si se anclara a la punta de una espada, y se quisiera averiguar si ésta se está moviendo lo suficientemente rápido como para provocar daño, la velocidad que importa es la del eje perpendicular a su filo, no si se está moviendo en la dirección X, Y o Z del entorno. Este dato es proporcionado por una función la API similar a la usada para obtener la velocidad global.
- Comparando la diferencia entre la velocidad en dos instantes, y midiendo la diferencia de tiempo mediante la librería adecuada de Python, es fácil calcular también las aceleraciones. El acelerómetro devuelve su aceleración global, **A**, sus componentes en el espacio global, **Ax**, **Ay** y **Az**, y sus componentes en el espacio local del acelerómetro, **Apx**, **Apy** y **Apz**.

Con todos estos valores disponibles, es fácil para el desarrollador programar eventos que se activen cuando el acelerómetro supera cierta velocidad o aceleración, en la componente que sea necesaria. Para que el script funcione y los valores se registren correctamente, aun así, el acelerómetro necesita tener un tipo físico definido. Esto significa que Blender debe considerarlo un objeto sólido, lo cual es un inconveniente para un objeto que solo sirve de medidor: sería muy contraproducente que colisionara o afectara de cualquier forma a otros objetos. Afortunadamente, Blender permite asignar un tipo físico a un objeto y, a la vez, activar una opción de intangibilidad, el modo fantasma, de forma que el resto de objetos lo atraviesen sin más.

Si el acelerómetro mide velocidad y aceleración, la **cuerda** mide distancias. Se compone de dos pequeñas pirámides, o extremos, de física igual a la del acelerómetro. Utiliza una de las funciones que Blender permite realizar a los objetos del juego: detectar objetos que tengan una propiedad lógica determinada. Uno de los dos extremos tendrá la propiedad en cuestión, cuyo valor y tipo es indiferente pero cuyo nombre debe ser único en la escena. El

otro extremo tendrá implementado el detector, que hará que una propiedad booleana, **Far**, cambie de *True* a *False* si la distancia entre los dos objetos es menor o mayor que una distancia determinada, que puede ser fácilmente editable por el desarrollador.

Las aplicaciones de la cuerda son muchas: se puede asociar uno de los extremos a un pie del personaje y el otro a una sección del suelo, por ejemplo una baldosa trampa que active un fuego cuando el personaje la pise. O se puede asociar cada extremo a una articulación *Hand* del esqueleto Kinect, de forma que se abra el menú de pausa cuando el jugador choque las manos. O puede medirse cuánto estira los brazos el jugador poniendo un extremo en la muñeca y otro en el hombro. Incluso se podrían crear varios extremos con la misma propiedad, de forma que el extremo detector reaccione a cualquiera de ellos de igual manera. En la Figura 18 se muestran los dos tipos de objetos.

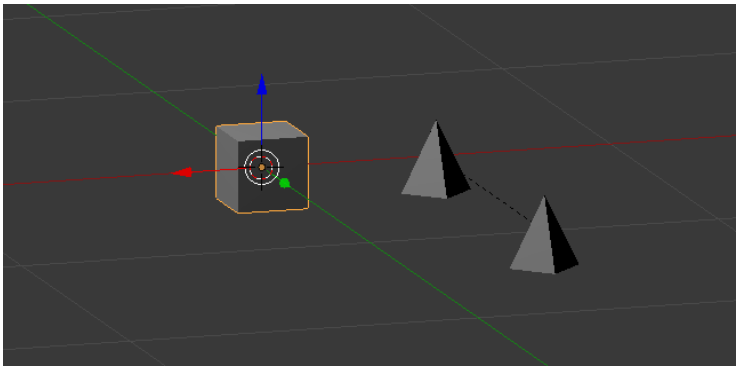

Figura 18- Acelerómetro y cuerda

3.2.4.2. Amplificación de movimientos

Aún hay un objeto auxiliar y un tipo de esqueleto presentes en el *add-on* cuya función no se ha explicado: los amplificadores y el esqueleto estático. Estos dos elementos se han desarrollado con vistas al objetivo final de este proyecto, que es la rehabilitación de personas con movilidad reducida. Eso significa que muchas veces los movimientos que el jugador pueda realizar serán insuficientes para que el juego los reconozca como comandos, o por lo menos mucho más tenues que los de un usuario sin restricciones físicas. Puesto que el objetivo es proporcionarles, no solo ejercicio, sino diversión y sensación de autosuficiencia, los juegos deben poder ser adaptables a cualquier rango de movimientos que el usuario pueda hacer.

Para eso se ha creado el objeto **amplificador**. Su función es detectar movimientos ligeros de una extremidad del jugador (mano, pie, cabeza o torso) y reinterpretarlos como un desplazamiento mucho mayor. Este proceso está dividido en dos partes: la amplificación propiamente dicha, y su traslación a un esqueleto de personaje.

La amplificación se realiza midiendo la distancia que se ha desplazado una extremidad desde su punto de reposo, y señalando el punto del espacio que equivale a ese desplazamiento multiplicado por un factor. El factor tendrá tres componentes, que permitirán aplicar desplazamientos distintos e independientes en los ejes X, Y y Z. Tanto estos valores como el punto de reposo son configurables y dependientes de cada usuario individual. Más adelante se detallará la forma en la que ambos son identificados y establecidos.

A continuación se representa un ejemplo simulado en el editor de Blender: en la parte superior de la Figura 19 puede verse la muñeca y la mano de un esqueleto que tiene el brazo extendido hacia delante. Se supondrá que ésa es su **posición de reposo**. La esfera roja representa este punto, que se encuentra exactamente en el extremo del hueso de la mano y que queda fijo respecto al conjunto del esqueleto. La esfera verde representa el **punto real** en el que se encuentra la mano en cada instante, que en este momento es el mismo que el de reposo. La esfera amarilla representa el **movimiento amplificado**: como en este momento la esfera verde y la roja son concéntricas, la diferencia entre ambos puntos es cero, por lo que el desplazamiento amplificado es cero también. Las tres esferas se encuentran en el mismo punto.

En la parte inferior de la figura, el brazo no se ha movido pero la mano se ha girado ligeramente hacia arriba. Hay que mencionar que, en aras de la simplicidad, la figura representa únicamente el plano YZ, por lo que el desplazamiento y la amplificación en el eje X son irrelevantes para el ejemplo. El factor de multiplicación en este caso es de 3 en el eje Y, el horizontal, y de 5 en el eje vertical Z. Las líneas azules ilustran cómo las distancias entre las esferas roja y verde han sido multiplicadas por estos valores para dar con la posición del punto amplificado, representado por la esfera amarilla.

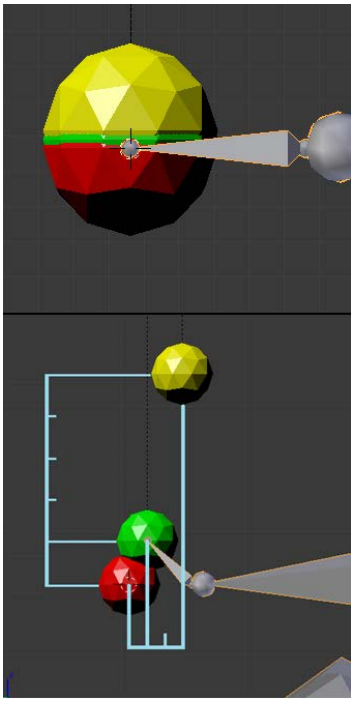

Figura 19 - Esquema de la amplificación de movimiento

Este ejemplo ha servido para ilustrar el concepto usado de la amplificación, sin mostrar su realización práctica. La mejor forma de obtener el efecto mostrado en la figura es utilizando una **armadura** como objeto amplificador, puesto que es muy sencillo asignar restricciones a huesos para que sigan el movimiento de la extremidad. Las restricciones permiten, además, aplicar un factor multiplicativo a los desplazamientos de los huesos, pero con una salvedad: el factor sólo puede ser un decimal entre 0 y 1. Los potenciales usuarios de estos juegos podrán necesitar, en cambio, un factor de hasta 25, ya que algunos sólo

podrán realizar movimientos realmente tenues con la mano o el pie, y con ellos deberán cubrir el alcance entero de un brazo o una pierna.

Esta amplificación puede lograrse combinando los desplazamientos de varios huesos, gracias a una opción en el tipo de desplazamiento forzado por las restricciones.

La posición de un hueso se puede expresar de dos formas diferentes: en el espacio **global** y en su espacio **local**, es decir, en relación a su hueso padre. Las restricciones de posición pueden operar según esta distinción: se puede ordenar a un hueso que copie la posición global de otro hueso modelo, en cuyo caso ambos estarán en el mismo punto del espacio. Pero si se ordena a un hueso que copie, en su espacio local, la posición local de otro, el resultado es que el hueso se aplicará el mismo desplazamiento respecto a su hueso padre que el modelo tiene respecto a su propio padre.

Sabiendo esto, el objeto amplificador puede alcanzar el factor deseado creando una **cadena** de huesos, cada uno hijo del anterior. Si a cada hueso se le aplica una restricción, copiando en su espacio local la posición local de su padre, entonces cada hueso tendrá un desplazamiento local igual al desplazamiento global del primer hueso de la cadena, que será el que siga el movimiento de la extremidad en el espacio global.

En la Figura 20 se puede ver el resultado. En la posición de reposo, todos los huesos del amplificador se encontrarían superpuestos unos sobre otros. El primer hueso de la cadena, mediante una restricción de localización, imita el movimiento de la mano. El lugar donde se emplaza la armadura, marcado en la imagen por los dos ejes, es el punto que representaba la esfera roja, la posición de reposo de la mano, mientras que el movimiento del primer hueso es el de la esfera verde. Se puede observar que el segundo hueso del amplificador está tan alejado del primero como el primero del punto de reposo, y que el tercero está desplazado del segundo esta misma distancia etc. Y así hasta el hueso número 25, cuya posición no está abarcada por la figura.

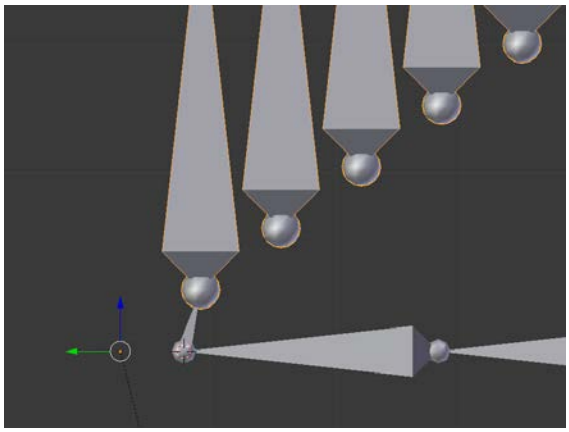

Figura 20 - Objeto amplificador

De esta forma, hay un hueso que representa cada factor entero de amplificación. Sin embargo, será necesario utilizar factores no enteros, y además deberá haber un factor diferente en cada eje XYZ. Para solucionar este problema se ha añadido un hueso más a la cadena: éste será el hueso que represente verdaderamente el desplazamiento amplificado. Tendrá como padre el primer hueso de la cadena, el que copia el movimiento de la mano, y

tres restricciones que le harán imitar la posición del hueso número 25 en el espacio global: una para la posición en el eje X, otra para el eje Y, y otra para el eje Z. Controlando el factor de aplicación de estas restricciones, el hueso se moverá entre la posición del hueso 1 (con la restricción a 0) y la del 25 (con la restricción a 1), determinando así los verdaderos factores de amplificación de esta extremidad.

Como se ha mencionado, tanto los factores de amplificación como las posiciones de reposo son distintas para cada usuario. La mejor solución para poder gestionarlos es que, la primera vez que utilice el sistema, cada jugador cree un **perfil de usuario** que quede guardado en un archivo, en el que queden reflejados estos dos valores.

Para el proceso de creación y guardado de estos archivos, lo ideal es utilizar un esqueleto de tipo combinado. De esta forma no habrá posibilidad de error al reproducir la pose del jugador. Con esta finalidad se ha añadido al esqueleto la propiedad "File", que indicará la ubicación del archivo en el que guardar o leer los datos, y se han puesto a disposición del desarrollador cuatro scripts de Python adicionales, que deben ser llamados durante la configuración del perfil de usuario:

1. **Registro de la posición de reposo**. Simplemente, en el momento en el que este script es invocado, guarda en el fichero las posiciones actuales de todas las articulaciones.
2. **Actualización de la posición máxima.** Este script debe ser llamado periódicamente en un determinado intervalo de tiempo: cada vez que se le invoque, medirá la posición de cada articulación y la comparará con su posición de reposo. Actualizará en la memoria temporal de Blender el valor más alto que se haya registrado a lo largo de las llamadas sucesivas a esta función, pero no la guardará en el fichero.
3. **Registro de la posición máxima**. Ésta es la función a la que se llama al final de la creación del perfil. En este momento toma los valores más altos de las diferencias de localización calculadas por la función 2, y los guarda en el fichero indicado en "File". Estos valores representan el desplazamiento máximo posible de esa articulación para ese usuario.
4. **Ajuste de los factores de amplificación**. Esta función debe llamarse al iniciar una escena en la que se hayan incluido amplificadores. Su trabajo es leer el fichero, recolocar todos los amplificadores de la escena para que se correspondan con su pose de reposo, y modificar los factores de ampliación en las restricciones.

La dependencia de los amplificadores de las posiciones de reposo hace que las variaciones en la colocación del jugador no sean bienvenidas: ya que las coordenadas se expresan respecto al Kinect y sus ejes, si en una sesión de juego el usuario está girado un par de grados respecto a la vez que realizó la configuración, la diferencia en la colocación de las articulaciones puede hacer muy incómodo el uso de los amplificadores.

Para solucionar esto se ha creado un nuevo tipo de esqueleto que esté orientado siempre en la misma dirección: el **Esqueleto Estático**. Este esqueleto funciona igual que el de rotación, con la diferencia de que el movimiento del hueso HipCenterRot, es decir la orientación global del usuario, no afecta al resto de huesos. El hecho de que el middleware

transmita las rotaciones jerárquicas hace que este esqueleto siempre esté mirando hacia delante, por lo que las posiciones de las extremidades no variarán si el jugador se gira sobre sí mismo. Para que esto sea efectivo, por supuesto, el movimiento global debe estar desactivado.

Otra particularidad de este esqueleto es que, pese a la obligación de usar las rotaciones como referencia, para que los amplificadores funcionen correctamente es vital representar la posición de las articulaciones de forma exacta, por lo que una longitud fija de los huesos lo desbarataría todo. La solución es no conectar los huesos entre sí, pero sí heredar las rotaciones. Leyendo del archivo la posición de reposo de cada hueso, se puede averiguar a qué distancia están las articulaciones, y por lo tanto los otros huesos, unos de otros. Sólo es necesario leer este valor una vez, puesto que la longitud de los huesos del jugador permanece constante. Al iniciar la escena, un script de Python realizará la lectura y moverá los huesos para imitar la estructura ósea del jugador. Esta reconstrucción, junto a la recepción de las rotaciones, permite reproducir con certeza las posiciones de las articulaciones y por lo tanto usar los amplificadores con comodidad.

Realizada la amplificación, falta aplicarla al movimiento del personaje protagonista. Hay que tener en cuenta que, independientemente de los amplificadores, la pose del esqueleto receptor sigue siendo la del usuario, y ése no es el efecto buscado. La solución consiste en variar las restricciones que se aplican al esqueleto del personaje: en vez de imitar la rotación de los huesos del receptor, hay que hacer que sus extremidades sigan al hueso que marca el desplazamiento amplificado. Existe una restricción que lo facilita, la **cinemática inversa** o *Inverse Kinematics*, que aplicada a una extremidad permite ordenarle que apunte hacia el objeto que se desee, y que "lleve tras de sí" a tantos de sus huesos padres como se le indique. La parte derecha de la Figura 21 ilustra este proceso: la esfera roja, de nuevo, representa el punto de reposo del jugador (se encuentra en silla de ruedas, por lo que la mano queda a la altura de la cintura). La esfera verde representa el desplazamiento vertical de su mano, que en este caso se considera el máximo que puede realizar. La esfera amarilla es el punto amplificado, y hacia ella se ha ordenado que apunte la mano del personaje. Se ha ajustado la restricción para que afecte no sólo a la mano, sino también al antebrazo y al húmero. En este caso se puede ver que la esfera queda dentro del alcance del personaje, por lo que el brazo se encuentra ligeramente flexionado. Si la amplificación fuera mayor, el personaje estiraría completamente el brazo para intentar alcanzar la esfera.

Este ejemplo revela un aparente obstáculo del sistema de amplificación: como se puede observar, el punto de reposo queda hacia la mitad del torso del personaje. Si el usuario está en una silla de ruedas, esto podría significar que el personaje no pueda bajar las manos más allá de ese punto, lo cual es obviamente un problema.

La solución es añadir un *offset* (una diferencia de posición que desplace su punto de referencia) a la copia de movimientos de los huesos del amplificador. De esta forma, colocando el amplificador en la posición de reposo del personaje, pero añadiendo la posición de reposo del jugador como *offset*, se puede lograr que el punto más bajo del alcance del jugador sea el punto más bajo del alcance del personaje.

Se puede ver esto puesto en práctica en la mitad izquierda de la Figura 21. Hay una bola roja y una bola verde en los puntos de reposo real y de la localización actual real de la mano del usuario, y una copia de cada una en los puntos correspondientes en el alcance del avatar, junto a su cintura. En la imagen izquierda superior, el usuario tiene la mano apuntando ligeramente hacia arriba (véase la pequeña diferencia entre la bola roja y la bola verde) lo cual provoca que el avatar sube su mano hacia el punto de la bola amarilla. En la imagen izquierda inferior, el usuario está descansando (la bola roja y la verde coinciden) y el avatar ha bajado la mano completamente a su punto de descanso. En la imagen de la derecha, se ve la correspondencia del movimiento de la mano del usuario (en la foto) con el movimiento del avatar.

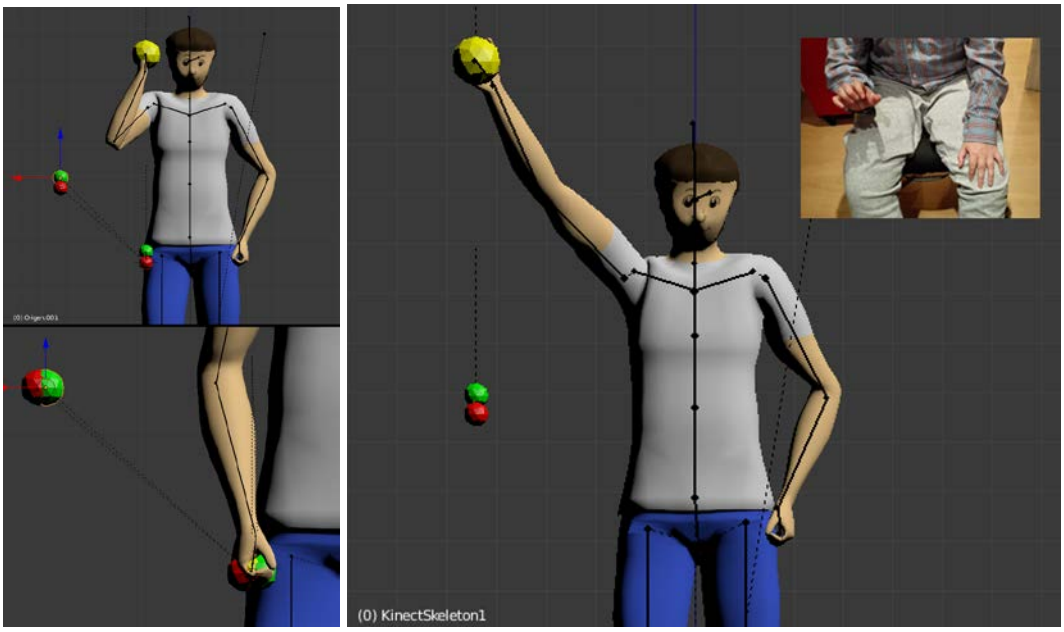

Figura 21 - Amplificador con offset *y personaje con cinemática inversa*

3.2.5. Otros receptores

En el punto 3.1.3 se habló de los dos *middlewares* adicionales incluidos en el Chiro: el HDM, para Oculus Rift, y el PhoneWindow, para dispositivos Android. Cada uno de estos *middlewares* forma parte de un sistema de comunicación que, por supuesto, incluye sus propios *add-ons* receptores en Blender.

Puesto que el objetivo de la realidad virtual es sumergir al jugador en el juego, el elemento clave dentro del entorno de Blender es una **cámara** en primera persona. Ése es el objeto receptor creado por el *add-on* de HMD: una nueva cámara activa para la escena, que 60 veces por segundo llama a una función que envía una petición al *middleware* y recibe a cambio un cuaternión: esa **rotación** es aplicada a la cámara en tiempo real y, como resultado, la imagen creada por el Game Engine se corresponde exactamente con los movimientos de la cabeza del jugador.

La imagen captada por esta cámara debe ser enviada al casco para poder ser visualizada por el jugador, pero el *add-on* no es responsable de esta parte: es el propio *middleware* el que captura la imagen de la pantalla en la que se encuentra el juego. Este sistema, sin

embargo, impone dos condiciones a la imagen para garantizar la inmersión total: debe mostrarse a pantalla completa, y debe ser **estereoscópica**. El motivo de esto es que el jugador debe ver dos imágenes distintas, una para cada ojo, y deben ocupar totalmente su campo de visión. Ambas, por suerte, son opciones básicas de visualización ofrecidas por el entorno de Blender.

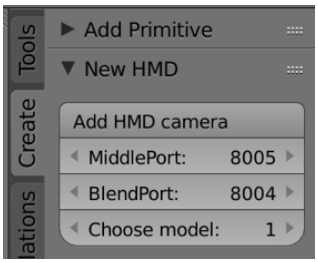

Figura 22 - Interfaz creada por el add-on *HMD*

El objeto receptor creado por el *add-on* para Android es, de nuevo, una armadura, pero compuesta por un solo hueso, ya que sólo se recibe una rotación: la del móvil. Para representar este objeto, el hueso tiene asociado un prisma, invisible e intangible durante la ejecución del juego, que imita la forma de un ***smartphone***, tal como se aprecia en la Figura 23.

La armadura abre la comunicación con el *middleware* al inicio de cada escena, y 30 veces por segundo recibe el cuaternión que describe la orientación del teléfono. Aplicando este cuaternión al hueso receptor, el prisma imita el movimiento del móvil. Con una salvedad: se dijo anteriormente que el móvil tomaba como referencia el polo norte magnético, pero esa referencia no tiene ninguna utilidad en el entorno del juego.

Por eso se ha añadido al receptor una función que permite, en el instante en el que se activa, **recalibrar** su posición. El usuario debe colocar el dispositivo sobre una superficie plana, apuntando hacia la pantalla del ordenador. El juego considerará que el cuaternión que reciba en ese instante representa a un móvil descansando en esta posición. Sabiendo eso, reajustará la posición global del objeto para que imite la orientación asumida del dispositivo. Si la calibración es correcta, desde ese momento, la rotación del objeto imitará perfectamente la del dispositivo real.

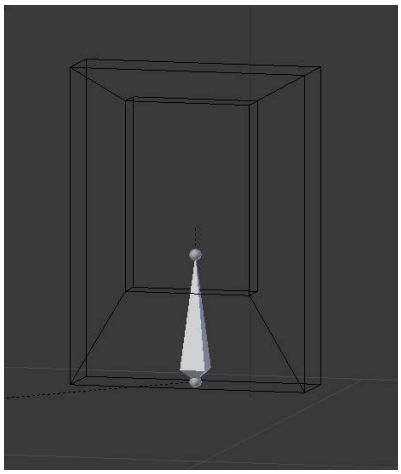

Figura 23 - Objeto receptor de dispositivos Android

3.3. Entorno de juegos para rehabilitación

Hasta ahora se ha descrito la implementación de la base técnica necesaria para que Blender pueda incorporar controles por Kinect u otras interfaces naturales. Aunque esto haya constituido el grueso del proyecto, no deja de ser una fase de preparación necesaria para poder realizar el trabajo principal: el desarrollo de los videojuegos para la rehabilitación de personas con discapacidad.

Parte del plan de trabajo del CITSEM para los próximos semestres incluye investigar la creación de un solo videojuego que englobe en sí todos los ejercicios necesarios para la rehabilitación. Se trataría de un juego perteneciente al género de aventuras o plataformas, que dependa para el progreso del jugador de la superación de distintos obstáculos basados en el ejercicio físico, pero motivados por el entorno y la historia del juego, de forma que el jugador no los reconozca como ejercicios clínicos. Estas pruebas deberán ajustarse automáticamente según la capacidad y las necesidades de los pacientes que lo jueguen. Incluso la propia narrativa del juego, idealmente, debería poder cambiar y adaptarse según el jugador.

El proyecto aquí presentado ha incluido, no sólo la implementación del sistema de control, sino también una significativa colaboración en el desarrollo de este juego, desde su diseño hasta la implementación y pruebas con usuarios de varios de los ejercicios. A este proyecto se le ha dado el nombre provisional de “Blexer”.

Pero antes incluso de empezar el diseño de Blexer, se desarrolló un juego corto, nada más completar la versión más primitiva del *add-on* que funcionaba con KinectOSC. Se trata de una demo que pretendía servir de campo de pruebas para integrar por primera vez un control por movimiento. Antes de detallar el proyecto del Blexer, se dedicará un apartado a hablar de esta demo.

3.3.1. La demo de trabajo

Se trata de un juego corto, con sólo dos niveles sencillos, de entre cinco y diez minutos de duración en total. Se desarrolló como demostración práctica de la implementación del *add-on*, tal como se encontraba en diciembre de 2014. Desde entonces, se ha actualizado conforme se desarrollaba el middleware y se modificaba el funcionamiento del receptor. Al iniciar el trabajo en Blexer se dio esta demo por finalizada, por lo que no hace uso de los amplificadores. Aunque tampoco haya cuerdas ni acelerómetros, la lógica de juego actúa en su lugar: de hecho, las cuerdas y los acelerómetros decidieron incluirse en el *add-on* tras comprobar en esta demo la clase de funciones que son necesarias para este sistema de control.

El título de la demo es “*The Legend of Blenda*”, en homenaje y parodia de la saga clásica de videojuegos “*The Legend of Zelda*” desarrollada por Nintendo [33]. Puesto que la demo está pensada sólo para exhibición del *add-on* y no para ser distribuida, se ha copiado la ambientación, el personaje principal y la mecánica de esta saga: en un entorno medieval, un guerrero élfico vestido de verde llamado Link avanza por campos y cuevas superando obstáculos y recogiendo diversos objetos, equipado principalmente con una espada. El

modelo del personaje y el de la espada se han obtenido de páginas de recursos 3D gestionadas por fans de estos juegos, mientras que el entorno se ha desarrollado muy toscamente a partir de cero.

La demo se desarrolla de la siguiente manera: tras una pantalla de título, se ofrece al usuario la opción de manejar a Link con el teclado o con la Kinect. Una vez seleccionada la opción "Kinect" aparece una cuenta atrás, permitiendo al usuario despejar la zona del teclado y la cámara y colocarse donde sea bien detectado. Es entonces cuando se inicia el juego propiamente dicho: Link aparece en un campo limitado por vallas de piedra, con una rampa que conduce a la entrada de una cueva, tapada con árboles. Es el escenario que aparece en la Figura 24.

El jugador debe hacer que Link suba por la rampa y, utilizando la espada, corte los árboles que bloquean la entrada a la cueva. Al entrar, tras una breve transición con la pantalla en negro, aparece el segundo nivel: una cueva circular, con un pozo en medio, y unas plataformas en la pared, tal como se ve en la Figura 25. El jugador debe sortear el pozo hasta llegar a un área tapada por árboles como los de antes. Al cortarlos se puede alcanzar un objeto: un paracaídas. El usuario puede entonces abrir el menú del inventario y seleccionar el objeto que equiparse. Sustituir la espada por el paracaídas permite saltar grandes alturas, lo que hace posible alcanzar las plataformas y alcanzar el punto más alto de la cueva. En ella hay un objeto en forma de corazón que, al ser recogido, marca el final del juego.

Figura 24 - Link en la escena 1 de la demo "The Legend of Blenda"

Éste es el guion diseñado para la demo, y su implementación planteó varios retos. La opción del teclado se implementó principalmente para el desarrollo, por si se quiere comprobar o mostrar algo pero resulta aparatoso utilizar el control por movimiento. El problema más apremiante era el más básico: mover a Link. Cualquier personaje de videojuego tiene que viajar distancias bastante grandes, mientras que el usuario no puede

moverse más allá del campo de visión de la Kinect. Era necesario que el jugador pudiera controlar por gestos el movimiento de Link. Se optó por hacerlo con los pies: un script de Python, al que se invoca regularmente, mide la diferencia de colocación entre los pies del usuario: si el pie derecho está adelantado, Link avanza, si está atrasado, retrocede. Si la separación entre los pies supera un margen, Link rotará hacia la derecha, si es menor, lo hará hacia la izquierda.

El esqueleto y la malla de Link no tienen tipo sólido, sino que están asociados a un cilindro que representa a todo el personaje en la simulación física del Blender Game Engine. A este cilindro es al que se le aplican los desplazamientos explicados arriba, Link simplemente se mueve con él. Esto hace necesario que el esqueleto Kinect (en este caso, uno de rotación) sea emparentado también con el cilindro. De esta forma, el esqueleto seguirá los movimientos del cilindro, y no se dará el caso de que Link esté mirando en una dirección mientras la representación del usuario en el juego esté orientada hacia otro lado.

Los huesos del brazo derecho de Link tienen restricciones para que imiten al brazo del jugador, es el brazo que lleva la espada o el paracaídas. Los objetos se usan agitando rápidamente la mano derecha: el mismo script que controlaba los pies mide en todo momento la velocidad de la mano, y en función del objeto equipado (indicado en una propiedad lógica del cilindro), hace saltar al personaje o alerta a todo el juego, mediante la transmisión de un mensaje, de que está dando un golpe con la espada. Los objetos destructibles, como los árboles, tienen un sensor que detecta la cercanía de la espada: si se recibe uno de estos mensajes con la espada a menos de un metro de distancia, el objeto se destruirá, dando la impresión de que Link lo ha echado abajo. Los objetos están siempre presentes en la escena, pero el que no esté equipado se hace invisible.

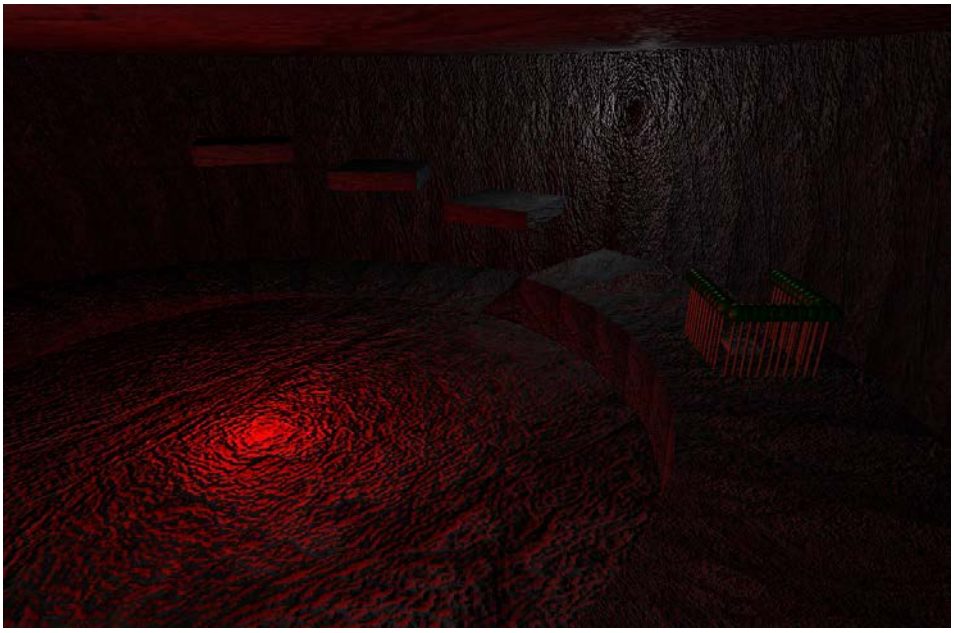

Figura 25 - El segundo nivel de la demo "The Legend of Blenda"

Pero sólo es el brazo lo que copia el movimiento del usuario. El resto del cuerpo ejecuta una animación, según esté quieto (se activa por defecto), andando (la activa el script que monitoriza los pies) o recogiendo un objeto (al chocar contra él). Las animaciones se han

grabado usando el propio *add-on*, asignándolas al esqueleto de Link. En el caso de la animación de recoger un objeto, que necesita el uso del brazo derecho, la lógica de juego permite desactivar temporalmente las restricciones que hacen que copie al receptor. Estas restricciones prevalecen sobre la animación, por lo que al desactivarlas el brazo ejecuta la misma animación que el resto del cuerpo, mientras que cuando están activadas la ignoran.

La selección de objetos se hace en la ventana del inventario, que muestra todos los objetos que el jugador puede equipar a Link. Como sustitución para el teclado, había que permitir realizar dos acciones: abrir esta ventana, y hacer una selección. Para abrirlo se mide la distancia entre la mano derecha y el hombro: se pretende evocar el gesto de envainar y desenvainar la espada a la espalda. El mismo script que monitoriza los pies mide la distancia entre estos dos puntos: si están cerca, añade el valor 1 a un contador, si no, lo reinicia a 0. Si el contador alcanza un valor determinado, es decir, si la mano se mantiene en esa posición un tiempo determinado, se pausa el nivel actual y se abre el inventario. Para seleccionar una opción, se ha introducido un cursor controlado por la mano derecha. Como no es posible hacer clic, y el estado de los puños puede suponer un método poco fiable, se opta por una solución parecida a la anterior: cuando el cursor se acerca a un botón, aparece un marco brillante que, mientras el cursor esté cerca, empieza a encoger. Si llega a cerrarse del todo, se considera que el botón está pulsado, y se envía un mensaje al cilindro para que cambie el objeto seleccionado. Puesto que el paracaídas no aparece hasta el segundo nivel, su botón no está disponible en el inventario hasta que el jugador lo recoja.

Lo único que falta por solucionar es mantener el control del Kinect estable, especialmente al cambiar de un nivel a otro y al perder la conexión. En el momento en el que se desarrolló esta demo, que la cámara perdiera de vista al jugador suponía que el juego dejara de recibir datos pero continuara abriendo sockets para intentarlo, por lo que su rendimiento se reducía gravemente. Para evitar esto, se hizo que cuando el receptor Kinect detecta una pérdida de datos, el juego se pausara y apareciera un menú, controlado, éste sí, por ratón y teclado. El menú ofrece tres opciones: cerrar el juego, retomarlo pero con el control tradicional, o reintentar la recepción de datos, en cuyo caso aparecerá de nuevo el contador para que el usuario se coloque en posición.

En realidad, este menú aparece obligatoriamente dos veces en el juego, solo que camuflado por otros menús: la primera vez es al iniciar el juego, cuando se supone que el usuario estará junto al teclado eligiendo el sistema de control. Seleccionar el Kinect controlará remotamente el menú de pausa para que haga aparecer el contador e inicie la recepción, mientras que elegir el teclado forzará la elección de continuar sin la Kinect. La elección que se haga en este momento se guardará en la memoria temporal del juego, y se repetirá al hacer la transición del nivel 1 al 2, momento en el que el receptor del nivel 2 debe ser configurado igual que en el 1.

El control de recepción lo realiza el propio esqueleto receptor: si se elige control por movimientos, el contador, al acabar, ordenará al esqueleto que abra un socket y empiece a recibir. Si la recepción falla, o si se elige el control tradicional, simplemente se ordenará al receptor que no vuelva a abrirlo, y que deje de llamar a la función de recepción.

Para mantener la información global sobre el juego (qué control utilizar, si el paracaídas está disponible o no), se ha añadido al juego una escena que está siempre superpuesta, con objetos invisibles que mantienen estos datos en sus propiedades lógicas y que mantienen al resto de objetos informados para que ejecuten una acción u otra.

Ésta es, a grandes rasgos, la implementación de un control por Kinect en un sistema de juego complejo. La demo se ha utilizado y exhibido repetidamente, tanto como para ilustrar la naturaleza del proyecto en una feria, ante visitantes, o ante otros colaboradores del proyecto que se hayan incorporado más tarde.

3.3.2. El proyecto "Blexer"

Blexer, el juego que se está desarrollando mediante la colaboración de varios alumnos en prácticas en el CITSEM, está planteado para cumplir con los estándares de cualquier juego que se lance comercialmente. Aunque su fin sea terapéutico, la filosofía del proyecto es que el juego resulte lo suficiente atrapante por sí mismo como para que un paciente de rehabilitación quiera jugarlo de forma regular, realizando así el ejercicio necesario sin tener la mente puesta en el esfuerzo físico ni los requisitos diarios, sino en el propio juego.

Para el desarrollo de Blexer se formó, en febrero de 2016, un equipo de tres personas: Yadira Peláez [34], Pablo Parra [35] y el autor de este proyecto. A lo largo de varias reuniones se discutió el tipo de juego que se quería realizar, cómo debería ser su jugabilidad, la historia en la que enmarcarlo, etc. Decidido esto, se hizo un reparto de trabajo: antes de julio se debería haber desarrollado una pequeña muestra de niveles del juego, para poder hacer pruebas con pacientes. Pablo y Yadira se dedicarían al desarrollo y modelado en Blender, y el autor se dedicaría a completar el desarrollo del *add-on* y el *middleware*, así como de orientarles en su manejo y ayudarles en el desarrollo.

Se decidió que el juego pertenecería al género de las plataformas 3D. Esta clase de juegos evolucionaron del género clásico de plataformas como "Super Mario Bros." o "Megaman", y su objetivo básico es avanzar a lo largo de una serie de niveles, superando obstáculos geográficos y personajes hostiles hasta alcanzar una meta, utilizando la acción de saltar como herramienta principal. En muchas iteraciones más modernas, los juegos de plataformas no sólo incluyen movimiento en los tres ejes, sino que permiten jugar sus niveles de forma no lineal, explorando zonas ocultas, eligiendo caminos alternativos y volviendo varias veces a la misma zona en busca de nuevos objetivos. Esto suele hacerse en servicio de una historia que el jugador va experimentando, que le puede ser narrada o en la que él puede influir, y de la cual va descubriendo más partes a medida que completa misiones en el juego. Además, la mecánica de juego puede llegar mucho más allá del clásico "saltar y atacar", incluyendo distintas habilidades que el personaje puede aprender u objetos que puede equiparse, cada uno orientado a la ejecución de un tipo distinto de tarea.

Un planteamiento similar a éste es el que se desea para el proyecto Blexer, en la escala que se pueda alcanzar con los medios disponibles. En las reuniones creativas del grupo se decidió que el juego estaría ambientado en un gran entorno con siete grandes zonas, habitadas tanto por personajes amigables como por enemigos, y que el jugador deberá guiar al personaje protagonista recorriendo y explorando estas zonas, cumpliendo misiones y

recibiendo nuevas habilidades como recompensa, que le permitirán acceder a nuevas áreas y misiones, hasta completar la historia.

En lo que respecta al argumento del juego, ésta es la sinopsis elaborada por los tres:

Un mundo poblado por animales inteligentes, que viven en pequeños poblados en una isla. El protagonista despierta en un lago, convertido en un diminuto renacuajo, sin recuerdos de quién es ni de cómo llegó allí. Con la ayuda de los habitantes del lugar, consigue salir del agua y desarrollar unas pequeñas extremidades. Investigando lo ocurrido, descubre que un misterioso villano está utilizando magia para robar los poderes evolutivos de todos los animales: la capacidad de salto de las ranas, la velocidad de los leopardos, el vuelo de los pájaros etc. El héroe debe emprender una aventura para rescatar a todos los animales en apuros, conseguir nuevas habilidades con las que evolucionar y derrotar a quien esté detrás de todo.

Se pretende que el personaje protagonista sea personalizable por el usuario: su sexo, su ropa, su color, incluso su especie: todo sería configurable. Se ha previsto una mecánica de evolución por la cual el personaje empiece siendo un pequeño animalillo pero, a medida que vaya consiguiendo nuevas habilidades y avanzando en el juego, su aspecto cambie y se desbloqueen nuevas opciones de personalización. En la Figura 26 se ve un boceto con una propuesta para sus distintos estados evolutivos: se puede apreciar que sus extremidades se van haciendo más largas y fuertes, permitiendo moverse más rápido, trepar y saltar, el desarrollo de pulgares permitiría usar objetos como armas o herramientas, e incluso hacia el final desarrollaría alas con las que volar o habilidades mágicas como telequinesis.

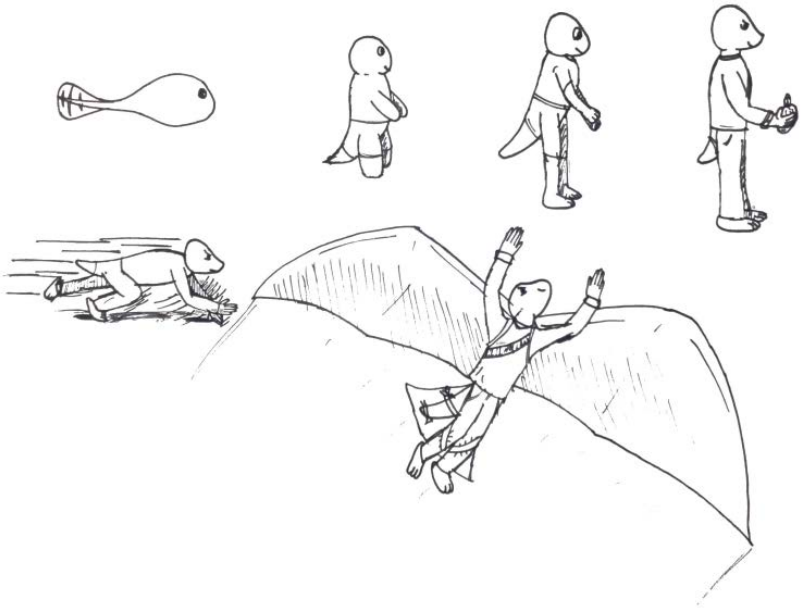

Figura 26 - Diseños del personaje principal [34]

Este último diseño, con alas, sintetiza bastante bien lo que se pretende ofrecer al jugador: una experiencia de mejora personal, con recompensas ofrecidas en la propia experiencia del juego como poder llegar a sitios antes inaccesibles o poder contemplar todo el mundo del

juego desde lo alto, una experiencia que ha sido sintetizada de forma muy satisfactoria en títulos ya existentes.

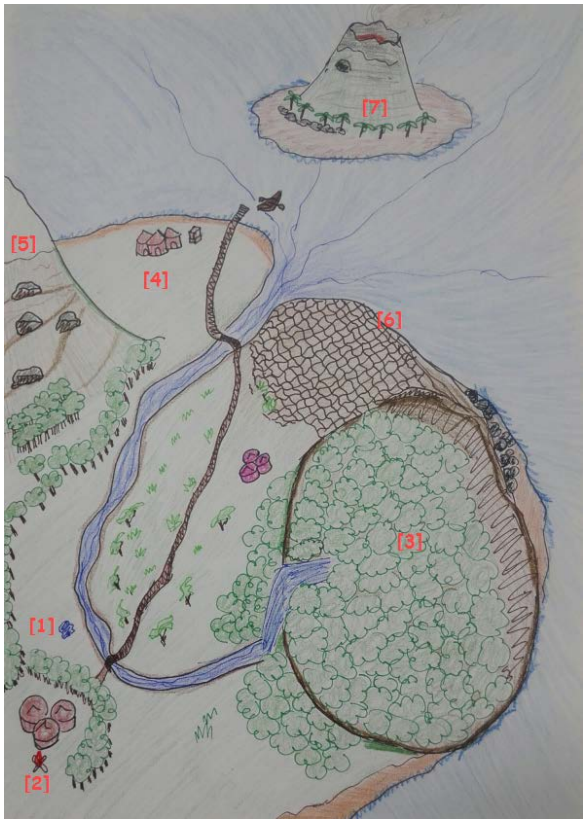

Figura 27 - Mapa del juego, por Yadira Peláez [34]

Pero lo más importante a corto plazo para el desarrollo del juego era la clase de ejercicios que el jugador tendría que hacer. En una de las primeras reuniones se hizo una lista de posibles acciones que permitan al personaje avanzar por los niveles o combatir contra enemigos. Estos ejercicios se dividen en tres categorías: los que utilizan sólo las manos (listados en la Tabla 1), los que utilizan también los pies (Tabla 2) y los que incluyen el torso (Tabla 3).

Tabla 1 - Ejercicios con las manos

Ejercicio	Descripción	Realización
Lanzar	El personaje debe lanzar un objeto, por ejemplo una piedra a un enemigo.	El usuario debe realizar la acción con una o ambas manos.
Espada	Blandir un arma, como una espada o un palo, que utilizar contra los enemigos.	El personaje imita los brazos del jugador.
Remar	Utilizar uno o dos remos para hacer avanzar una barca.	El usuario debe realizar la acción con las manos.
"Guacamole"	Utilizar un martillo para aporrear a enemigos al nivel del suelo.	El personaje imita los brazos del jugador.
Gancho	El personaje lanza un gancho, unido a una cuerda, a un saliente al que pueda agarrarse para trepar.	El usuario debe realizar el giro de muñeca o el movimiento del brazo.
Trepar	El personaje se agarra con ambos brazos a una cuerda por la que va trepando.	El jugador debe realizar con las manos el gesto de trepar.
Instrumentos	Manejo de instrumentos pequeños como pedernal para hacer fuego, una ganzúa para abrir cerraduras, etc.	El jugador debe realizar sutiles gestos con las manos.
Arco	El personaje debe tensar la cuerda, apuntar y lanzar.	El jugador debe imitar el gesto de disparar un arco, con el brazo izquierdo apuntando en la dirección deseada y la mano derecha realizando el movimiento de tensar la cuerda.
Nadar	El personaje se desplaza por el agua a estilo braza o crawl.	El jugador debe realizar el movimiento con los brazos.
Telekinesis	Hacia el final del juego el personaje aprende a mover objetos a distancia y hacerlos flotar.	Con el movimiento de un brazo, el jugador desplaza el objeto hacia los lados, arriba o abajo. Flexionando el brazo, atrae el objeto hacia sí. Estirándolo, lo repele.

Tabla 2 - Ejercicios que incorporan el uso de los pies

Ejercicio	Descripción	Realización
Salto	La acción básica en la mayoría de videojuegos.	El usuario debe levantar rápidamente uno o ambos pies en la dirección en que quiera saltar.
Fútbol	Mover o lanzar objetos golpeándolos con el pie.	El personaje imita los pies del jugador.
Escaleras	El personaje debe subir por un tramo de altísimos escalones.	El jugador debe levantar alternativamente uno y otro pie.
Sprint	El personaje echa a correr a toda velocidad. (Variante: a cuatro patas)	El jugador debe agitar alternativamente un pie y otro. (Variante: también las manos).
Piedras del río	Una serie de pequeñas plataformas. El jugador debe mover cuidadosamente los pies de una a otra.	El movimiento del pie levantado imita al pie correspondiente del jugador.
Baile	El personaje debe colocar los pies en determinadas casillas de una cuadrícula.	El personaje imita los pies del jugador.
Rocódromo	El personaje debe valerse de pies y manos para agarrarse a pequeñas piedras y subir una pared vertical.	La mano o pie que, en cada momento, no estén agarrados, imita los miembros del jugador.

Tabla 3 - Ejercicios que requieren el uso de todo el cuerpo

Ejercicio	Descripción	Realización
Coleteo	Al inicio del juego, el protagonista es un renacuajo sin extremidades, sumergido en un lago. Debe mover la cola para desplazarse.	Moviendo el tronco de un lado a otro repetidamente.
Surf	El protagonista debe dirigir una tabla de surf o de snowboard. Moviendo su peso de un lado a otro, controlará la dirección de avance.	Moviendo el tronco en la dirección deseada.
Equilibrismo	El personaje debe cruzar una cuerda floja. La inclinación del tronco debe contrarrestar el movimiento al andar, o de lo contrario caerá al vacío.	El movimiento sucesivo de los pies izquierdo y derecho determina el avance por la cuerda. El tronco del jugador controla la inclinación del personaje.
Arrastrarse	El protagonista debe avanzar por un estrecho túnel o conducto, o esconderse tras un obstáculo, impulsándose con los antebrazos.	El movimiento sucesivo de los antebrazos controla el avance. El tronco debe permanecer lo más inclinado posible.
Tirolina	El personaje se cuelga y se desliza por una cuerda tensa, esquivando obstáculos a los lados y por debajo.	El jugador debe mantener las manos juntas en alto para no soltarse y caer. Balanceando el tronco esquiva obstáculos laterales, y levantando las piernas esquiva obstáculos por debajo.
Lianas	El personaje se cuelga de una liana, balanceándose para coger impulso y agarrar la siguiente con las manos.	El jugador debe balancear el torso, mantener al menos una mano pegada al mismo para no soltarse, y alargar el otro brazo para alcanzar la siguiente liana.
Entrenamiento	Un personaje no controlable (NPC) realiza una serie de poses y movimientos que el protagonista debe imitar.	Se utiliza el cuerpo completo del jugador.
Esquivar	Los enemigos disparan una serie de proyectiles contra el protagonista, y éste debe apartar sus brazos, piernas, cabeza ¡o todo! de su trayectoria.	Se utiliza el cuerpo completo del jugador.
Volar	El personaje consigue unas alas con las que planear. La dirección en las que las incline determina la dirección del vuelo, batiéndolas obtiene un impulso hacia arriba y pegándolas al cuerpo cae.	Los brazos del jugador controlan el movimiento de las alas. Inclinando el tronco se inclina el personaje hacia delante o hacia atrás.

Con esta fase del diseño hecha, se procedió a intentar desarrollar individualmente algunos de los ejercicios. Este desarrollo fragmentado permitió a Yadira y a Pablo familiarizarse con la integración de la Kinect en Blender, e ir implementando poco a poco los controles con ayuda de las cuerdas y los acelerómetros. Al final del semestre, en junio de 2016, se tenían

ya cinco minijuegos funcionales que pudieron utilizarse por primera vez en pruebas con pacientes de rehabilitación, cuyos resultados se detallan más adelante y que incluían el uso de los amplificadores. Cada uno de ellos, además, contaba con una versión compatible con las gafas de realidad Oculus Rift, adaptados por José Zarco [25]. Estas cinco aplicaciones fueron:

- **El baile** (Yadira Peláez [34]). En el futuro juego habrá al menos una misión que requiera que el personaje ejecute una serie de pasos que se le indiquen, por ejemplo colocándose encima de una serie de baldosas y pisando la que se ilumine. En este minijuego, como sustituto provisional a pisarlas, las baldosas aparecen delante, detrás o a los lados del personaje, y el objetivo es simplemente tocarlas. En lugar de que las piernas imiten exactamente los movimientos del jugador, el esqueleto receptor tiene a su alrededor una serie de cuerdas que detectan si la separación (lateral o frontal) de los pies es suficiente. Si lo es, se activa una animación en la malla del personaje por la cual mueve el pie indicado hacia delante, detrás o un lado, tocando la baldosa. Se ha aprovechado también para experimentar con una variante del fútbol: las baldosas son aquí objetos sólidos que reaccionan de forma realista a ser golpeadas con el pie. Es notable que esto se haya conseguido porque dotar de solidez a un esqueleto o una malla, en Blender, puede ser muy problemático. La malla puede colisionar consigo misma o deformarse, provocando errores que descuadran el funcionamiento de todo el juego. La solución ha sido hacer la malla intangible pero asociar a sus pies dos objetos sólidos invisibles, que sí colisionan con las baldosas.

- **La escalera** (Pablo Parra [35]). El personaje tiene que trepar por una escalera de mano. Para esta versión se han ignorado los movimientos de los pies en favor de usar únicamente las manos. El funcionamiento es sencillo: una de las manos del personaje está agarrada a la escalera, la otra imita los movimientos del esqueleto receptor. Una cuerda junto a éste detecta cuándo la mano libre ha subido lo suficiente, haciendo que ésta pase a agarrarse al escalón, el personaje active una animación de subir, y sea la otra mano la que se libere y deba subirse. Se debe alternar el movimiento hacia arriba de cada brazo hasta llegar a lo alto de la escalera. El juego, además, cronometra el tiempo total que tarda cada jugador en subir. Se puede ver una captura de pantalla en la parte inferior derecha de la Figura 28.

- **La barca** (Pablo Parra [35]). El personaje protagonista está sentado en una barca, con un remo en cada mano. Ambos brazos imitan el movimiento del jugador: éste debe lanzar las dos manos alternativamente hacia adelante y hacia atrás para ser detectadas por dos cuerdas sucesivamente. Cada vez que se cumpla un ciclo de "cuerda delantera activada, cuerda trasera activada", la barca avanzará unos metros. El objetivo es llegar a una meta, también con el ritmo cronometrado. Aparece en la esquina superior izquierda de la Figura 28.

- **El "guacamole"** (Pablo Parra [35]). El protagonista empuña un enorme martillo en su mano derecha, y rodeado por cuatro agujeros de los que salen unos molestos topos, y se vuelven a esconder al cabo de unos segundos. El jugador debe mover el brazo

para alcanzar y golpear a los topos en los intervalos en los que están al descubierto. La colisión se detecta al tocarse la punta del martillo (un objeto sólido) con los topos, pero sólo se contará como golpe si el acelerómetro que hay en el mazo detecta una velocidad lo suficientemente alta. Los topos, al ser golpeados, hacen un gesto de dolor con efecto de sonido y vuelven a esconderse. El juego concede dos minutos para intentar golpear a 21 topos. Se puede ver la escena en la esquina superior derecha de la Figura 28.

- **El vuelo (Ignacio Gómez-Martinho)**. En este minijuego, representado en la esquina inferior izquierda de la Figura 28, se controla a una especie de pájaro que va avanzando siempre en la dirección a la que apunte su parte delantera. Ésta se puede girar hacia izquierda o derecha extendiendo los brazos como si fueran alas: una estructura de huesos dibuja dos rectas entre cada hombro y cada mano, y luego la bisectriz del ángulo entre ambas rectas marca, en el plano vertical, la dirección hacia la que girará el pájaro en su plano horizontal. Si el brazo derecho apunta hacia abajo mientras el izquierdo apunta hacia arriba, el pájaro girará hacia la derecha. Con la disposición contraria girará hacia la izquierda. Si ambos brazos están extendidos a la misma altura, seguirá recto. El movimiento del torso, por otra parte, controla la inclinación: si el jugador se inclina hacia detrás, el pájaro empezará a tomar altura, mientras que si se inclina hacia delante el pájaro descenderá en picado. Combinando ambos elementos se puede maniobrar el personaje por un entorno que representa un campo con un río, rocas y una montaña. No hay límite de tiempo ni un solo objetivo, pero se han dispuesto aros flotantes para que el jugador pruebe su dominio del control intentando pasar por su centro.

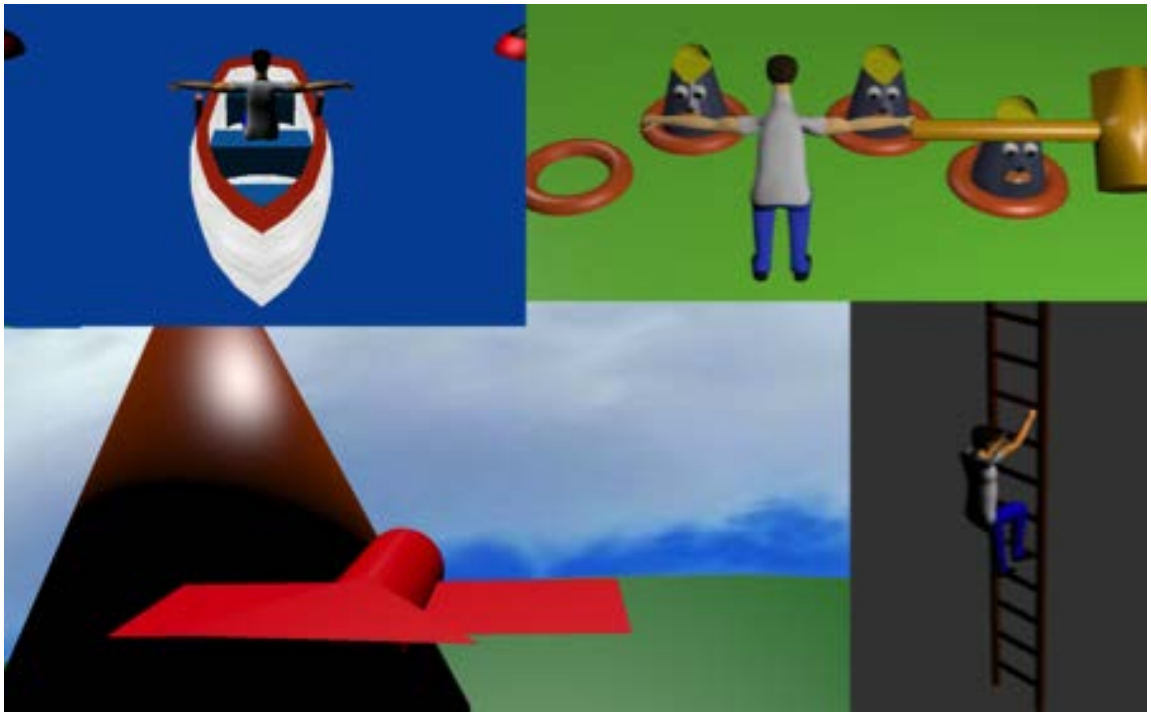

Figura 28 - Cuatro minijuegos desde arriba hacia abajo: La barca, "Guacamole", el vuelo, la escalera.

4. Pruebas realizadas

Tratándose de un proyecto de larga duración, ha dado tiempo a probar y evaluar su funcionamiento repetidas veces: en algunas ocasiones en demostraciones completas con sujetos invitados a probar los juegos, pero casi siempre mediante el uso diario y repetido del *middleware* y el *add-on*.

4.1. Observaciones generales

Lo primero que hay que recalcar es que, teniendo en cuenta la gran cantidad de datos que se deben procesar en tan poco tiempo, es sorprendente la fluidez con la que funcionan ambos extremos de la comunicación. Cuando se desarrolló la demo de "Blenda", el KinectOSC mandaba la información de posición cada vez que la Kinect actualizaba los datos: como resultado, la velocidad de la comunicación era inconstante y el juego se ejecutaba a una media de 12 fps (para videojuegos modernos se consideran deseables 60 fps, y 30 fps es el mínimo aceptable). Una vez codificado el *middleware* de la KinectWindow, éste mantiene en su memoria los datos más recientes disponibles sobre el jugador, y que es Blender quien envía una petición de datos a la que el *middleware* responde inmediatamente. Al ser Blender quien lleva la iniciativa en la comunicación, se ha experimentado una gran mejoría: a partir de esta versión, los juegos se han ejecutado invariablemente a unos sólidos 60 fps.

La traslación de los movimientos de la Kinect al Blender funciona muy bien: el esqueleto traduce a la perfección los datos que la Kinect le envía. Sin embargo, es la captación de estos datos lo que no siempre es perfecto. En condiciones extremas de luminosidad, especialmente en presencia de superficies reflectantes o en espacios iluminados directa o indirectamente por el sol, a la Kinect le cuesta realizar el escaneado del esqueleto. Esto se debe a que esta detección se realiza en base a la imagen de profundidad, que la Kinect construye utilizando su emisor de luz infrarroja. Si otras fuentes de luz interfieren con esta detección, la detección puede dar resultados aberrantes.

Sin embargo, incluso en condiciones óptimas suele revelarse la mayor deficiencia de la Kinect: la detección de manos y pies. Las articulaciones HandRight, HandLeft, FootRight y FootLeft no representan las puntas de estas extremidades, sino más bien un punto medio en su longitud. Se tratan, además, de huesos muy pequeños, en zonas muy heterogéneas del cuerpo que necesitan más precisión para observarse. Como resultado, la localización de estas articulaciones puede titilar en el tiempo, incluso con el jugador quieto. No es posible detectar con precisión la rotación de estos huesos sobre su propio eje, como tampoco lo es en el hueso de la cabeza. Incluso la detección del estado delas manos, abiertas o cerradas, es inconstante, y la propia SDK advierte de que sólo se garantiza su precisión si el jugador tiene las palmas apuntando hacia la cámara. Esta condición, claro, puede integrarse dentro de la lógica del juego, pero sigue siendo una restricción.

Este problema se agrava si el jugador tiene los brazos muy cerca del cuerpo: la posición ideal es de frente a la cámara, con las piernas rectas y los brazos a los lados. Una extremidad en escorzo puede hacer que la Kinect pierda la referencia de la posición de las articulaciones en esa zona.

Sin embargo, a pesar de todas estas limitaciones, hay que recalcar que rara vez un videojuego con control de movimientos va a necesitar este nivel de precisión, y la mayoría de las pruebas con el Blenda han demostrado que la detección era correcta y suficiente.

4.2. Pruebas realizadas con la demo "The Legend of Blenda"

Como primera prueba delante público, se mostró la demo en la feria "El Aprendiz de Ingeniero" que se celebró en la Escuela Técnica Superior de Ingeniería Agronómica, Alimentaria y de Biosistemas de la UPM del 22 al 26 de septiembre. La Feria recibió alrededor de 3000 estudiantes de 40 colegios madrileños para que participaran en setenta actividades prácticas relacionadas con las áreas de estudios de la UPM. Junto con otro proyecto en marcha, el Mokey, se dejó probar la aplicación por niños y adolescentes durante 5 mañanas.

De las demostraciones hechas con el Blenda, el balance es en general positivo: la mayoría de las personas que lo probaron entendieron inmediatamente, y sin ninguna explicación, lo que el juego requería de ellos: al ver una colina quisieron subir a ella, al ver árboles y tener una espada en la mano quisieron cortarlos, al ver un pozo intentaron por todos los medios no caer a él etc. Hay un solo inconveniente: el control del movimiento del personaje, utilizando el pie, resultó confuso para la mayoría de las personas que no fueran jugadores habituales y dedicados de videojuegos. En concreto, fue el hecho de controlar el giro con la separación de los pies lo que les resultó poco intuitivo. De cara al juegos para rehabilitación habrá que diseñar un sistema de control más directo.

4.3. Pruebas realizadas con el entorno "Blexer"

En junio de 2016, durante seis días, en el CITSEM, se expusieron los cinco minijuegos desarrollados del proyecto Blexer, y se invitó a probarlos a varios pacientes, una sección de ellos con distintas afecciones que resultaban en una movilidad reducida, y otra con movilidad completa para comparar resultados. Algunos de ellos eran adultos, pero la gran mayoría se trataba de niños de entre 5 y 12 años. El objetivo de estas pruebas era, por una parte, evaluar el interés de los usuarios potenciales en un juego de estas características, y por otro, comprobar que el control diseñado y la aplicación de los amplificadores funciona y se adapta correctamente.

El procedimiento era el siguiente: cada usuario pasaba los primeros dos minutos creando un perfil propio, con su posición inicial y su amplitud de movimientos, tal como se describe en el Anexo X. El archivo con estos datos era luego cargado en cada minijuego que probaban, adaptando así la intensidad y la colocación de los amplificadores. Tras probar todos los juegos, se les invitaba a rellenar un cuestionario para contar sus impresiones.

En todos los casos, los usuarios con movilidad completa que jugaban de pie obtuvieron resultados perfectos: la calibración se realizaba correctamente al primer intento, y los amplificadores se ajustaban de tal forma que sus movimientos eran perfectamente replicados por los protagonistas de los juegos. Tanto el juego de la escalera como el de la barca, los topos y el baile eran manejados con soltura y precisión, aunque no por ello con menos

esfuerzo físico, que es lo que se busca en este proyecto. El juego del pájaro presentó más dificultades, y la mayoría encontró su sistema de control muy lioso: el concepto de controlar la dirección mediante la inclinación de los brazos ya resultaba complejo, y al combinarlo con el movimiento del torso se volvía directamente fastidioso.

Con los usuarios en silla de ruedas, el proceso se complicaba mucho. A la Kinect le cuesta detectar personas sentadas, y a menudo confundía los reposabrazos de la silla con las manos y las ruedas con las piernas. Esto resultaba especialmente grave en los casos de niños pequeños sentados en sillas grandes, donde la Kinect fracasaba totalmente en reconocer el cuerpo del jugador. El "modo sentado" de la Kinect mejora ligeramente los resultados, pero sólo habilita la detección de los brazos y la cabeza, y los juegos necesitan el uso de distintas partes del cuerpo.

En la mayoría de los casos fue necesario repetir la creación del perfil varias veces, y probar distintas colocaciones entre la Kinect y el usuario, hasta que la detección ofrecía resultados aceptables. En muchas ocasiones no se detectaba la presencia de un error hasta después de la fase de configuración, cuando el protagonista del minijuego que estuviera jugando realizaba movimientos extraños, o la pose de reposo del jugador no se correspondía con la del personaje.

En la mayoría de los casos, repetir la configuración hasta dar con los valores correctos daba resultados: una vez hecho esto, los juegos respondían muy correctamente. Se puede destacar el caso de un chico de 15 años afectado por una distrofia muscular, cuyo movimiento vertical con la mano se limita a un giro de muñeca, y que sin embargo fue capaz de utilizarlo para superar el juego de la escalera sin ninguna dificultad.

Aún en los casos en los que la detección y la amplificación eran correctas, se ha observado que este sistema tiene limitaciones: la configuración inicial sólo detecta los valores máximos de sus movimientos en los ejes X, Y y Z. Esto ignora, por una parte, la naturaleza exacta de los movimientos que el jugador puede realizar: por ejemplo, en el juego de la barca, la mejor forma que tenía el chico antes mencionado de hacer un movimiento amplio de los brazos desde atrás hacia adelante era moviéndolos en diagonal, apoyándose en sus piernas. A pesar de que, utilizando un amplificador, sus manos recorrían la distancia necesaria para hacer avanzar la barca, el juego esperaba un movimiento en horizontal, y respondía a éste de forma irregular. También hay que tener en cuenta que, aunque un niño pueda alcanzar un máximo de movimiento durante la fase de configuración, no significa que sea capaz de realizarlo repetidamente: uno de los niños se cansó enseguida y pidió que se interrumpieran la mayoría de juegos antes de poder completarlos.

Hay varios apuntes que hacer sobre el minijuego del pájaro: más allá de la dificultad de calibrar adecuadamente los amplificadores y la posición de reposo, resultó necesario hacer ajustes a nivel de lógica de juego para que el control del torso funcionara correctamente. En concreto, parecía que los usuarios sentados y los que estaban de pie necesitaban configuraciones distintas. Esto revela la necesidad de ofrecer más opciones en la creación del perfil. Sin embargo, los usuarios que llegaron a poder probar una versión ajustada

correctamente dominaron el control razonablemente bien, y algunos demostraron verdadero entusiasmo por el juego.

Los cuestionarios posteriores, realizados por los pacientes tras probar los minijuegos, revelaron un alto grado de satisfacción con el proyecto, y un deseo de experiencias más activas: el juego mejor valorado ha sido el de los topos, el único con un componente de acción. Los demás eran sólo pruebas mecánicas, con el único objetivo de desplazarse de una localización a otra. Es de esperar que estos pacientes se muestren interesados en el juego completo, una vez desarrollado todo su entorno y sus objetivos a cumplir.

5. Conclusiones

El objetivo de este proyecto ha sido desde el principio posibilitar la comunicación entre Blender y la Kinect. El trabajo empezó estudiando las posibilidades de Blender para ejecutar código Python, utilizando primero un programa externo como apoyo y modelo para codificar el receptor, y luego desarrollando un middleware desde cero. Actualmente, la comunicación entre ambos sistemas funciona de forma estable, y el sistema provee formas de enviar y recibir la información del esqueleto en todas sus formas posibles: rotación, posición, en el espacio, en la imagen, etc.

El desarrollo de la demo "*The Legend of Blenda*" demostró que este *add-on* ofrece herramientas suficientes para desarrollar un juego complejo que utilice el control por movimientos de la Kinect. El reto, como se comprobó al ponerlo a disposición de sujetos de pruebas, será implementar estos controles de formas intuitivas y cómodas para el jugador.

A mediados de 2015, el proyecto se expandió y empezó a entrelazarse con otros trabajos, como la compatibilidad de Blender con el Oculus Rift. Como respuesta a esto, se desarrolló el contenedor Chiro para poder agrupar todos los módulos que se comunicarán con los juegos de Blender. Entre ellos está la pestaña de dispositivos móviles, cuyo desarrollo se investigó durante unos meses hasta que se logró su funcionamiento básico. Sin embargo, a la luz de la integración del visor Oculus Rift por José Zarco [25], se evaluó como no prioritario y se dejó en un estado funcional muy primitivo.

La formación de un grupo de desarrollo para el proyecto Blexer hizo necesaria la simplificación de las herramientas para Blender, y esto llevó a la creación de los objetos auxiliares de control (cuerdas y acelerómetros). Su uso ha resultado muy sencillo y efectivo. A esto siguió la implementación de los amplificadores y los perfiles de usuario, cuya efectividad se probó en los cinco minijuegos probados con jugadores reales.

El desarrollo de los primeros ejercicios de Blexer, y sus pruebas con pacientes, han revelado las mayores limitaciones de todo este sistema: la precisión en la detección, especialmente en usuarios con limitaciones motrices. También se observó una disonancia entre la función de los amplificadores y las necesidades del jugador: éstos multiplican el desplazamiento de las articulaciones en los ejes X, Y o Z, cuando las capacidades de movimiento del jugador pueden llegar a necesitar descriptores mucho más complejos.

6. Trabajo futuro

La naturaleza de este proyecto hace que siempre se estén pensando en líneas de trabajo a seguir a largo plazo: el desarrollo completo del proyecto Blexer, sin ir más lejos. Se ha mencionado varias veces el deseo de integrar un sistema de *feedback* que permita al juego reconocer el estado de ánimo o el esfuerzo físico del usuario, y adaptar el juego en consecuencia: por ejemplo, si se lleva mucho tiempo ejercitando los brazos, redirigir al jugador de forma que se dirija a una zona que requiera el uso de las piernas, o simplemente una generación dinámica de enemigos especializados. Esta información se podría recoger mediante otra clase de interfaces naturales, como cardiómetros o detección facial.

La detección facial es una característica de la Kinect que ha quedado sin explorar en este proyecto: permitiría un reconocimiento de las emociones del jugador, y supondría un sistema de control complementario al del esqueleto, por ejemplo, controlando un personaje mediante el movimiento de la boca o el guiño de los ojos.

Pero, a corto plazo, hay muchos aspectos que pulir de las soluciones ya implementadas. El más urgente seguramente sea el de la calidad de la detección. La opción más evidente es comenzar el desarrollo de una nueva versión del middleware, que sea compatible con la Kinect 2.0. Ésta es una revisión del sensor Kinect, comercializada a finales de 2013 como accesorio para la consola Xbox One, sucesora de la Xbox 360. Su mayor ventana es una mayor resolución del sensor, alcanzando calidad Full HD. Esto permite una detección del esqueleto más exacta, incluyendo cinco articulaciones adicionales: una en el cuello y dos en cada mano, que son precisamente las zonas que peor relación precisión/importancia tienen en el modelo original de Kinect.

Figura 29 - El sensor Kinect 2.0

Sin embargo, hay posibilidades que tal vez permitan mejorar la detección en la Kinect actual, tales como el modo sentado o una configuración especial para usuarios cercanos al sensor. Deberán investigarse estas opciones y, si resultara conveniente, habilitarlas en el middleware y el *add-on*. Después de todo, este proyecto debería poder llegar al mayor número posible de pacientes, y para ello procede garantizar la compatibilidad con el mayor número posible de dispositivos – sobre todo teniendo en cuenta que la versión de Xbox 360 lleva más tiempo en el mercado y que su presencia entre el público es, por lo tanto, mayor. El problema de la precisión en la detección será más difícil de resolver.

También se demostró en las pruebas que el modelo actual de amplificación es un paso en la dirección correcta, pero insuficiente. En futuras continuaciones de este proyecto, deberá idearse una forma alternativa de adaptar los controles básicos de un juego a cada usuario.

Por último, hay pequeños inconvenientes técnicos que resolver en el sistema actual, que impiden que funcione tal y como se ha diseñado. Siendo todos de poca importancia, seguramente el más prioritario sea la imposibilidad de ejecutar el *add-on* con los juegos en modo autónomo. Es decir, el receptor no funciona si el juego se lanza desde su propio archivo ejecutable. Esto es así porque el add-on concentra en un mismo archivo el código destinado a modificar la interfaz de Blender y el código que deben ejecutar los juegos. Éstos, al leer el archivo, detectan partes de código que no pueden procesar y se produce un error. Deberá encontrarse la forma de poder separar estos dos tipos de scripts, de forma que ambos se incorporen al proyecto en las formas necesarias, sin que el desarrollador tenga que importar los distintos fragmentos de código de forma manual y separada.

7. Referencias

[1] M. Eckert, I. Gómez-Martinho, J. Meneses y J. F. Martínez Ortega, «A Multi Functional Plug-in for Exergames,» de *IEEE International Symposium on Consumer Electronics (ISCE)*, Madrid, 2015.

[2] M. Eckert, I. Gómez-Martinho, J. Meneses y J. F. Martínez Ortega, «A modular middleware approach for exergaming,» de *International Conference on Consumer Electronics (ICCE)*, Berlin, 2016.

[3] Wikipedia, «Wiimote,» Nintendo, 11 Julio 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Wiimote. [Último acceso: 13 Julio 2016].

[4] Wikipedia, «PlayStation Move,» Sony, 16 Mayo 2016. [En línea]. Available: https://es.wikipedia.org/wiki/PlayStation_Move. [Último acceso: 13 Julio 2016].

[5] Wikipedia, «Eye Toy,» Sony, 24 Abril 2016. [En línea]. Available: https://es.wikipedia.org/wiki/EyeToy. [Último acceso: 13 Julio 2016].

[6] Microsoft, «Kinect - Desarollo de Aplicaciones,» Microsoft, 2016. [En línea]. Available: https://developer.microsoft.com/es-es/windows/kinect. [Último acceso: 13 Julio 2016].

[7] UPM, «Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM),» 2016. [En línea]. Available: https://www.citsem.upm.es.

[8] «Blender.org,» 2016. [En línea]. Available: https://www.blender.org/. [Último acceso: 13 Julio 2016].

[9] Oculus, «Oculus,» Oculus V.R, 2016. [En línea]. Available: https://www.oculus.com/. [Último acceso: 13 Julio 2016].

[10] M. López García, «Desarrollo del Teclado Virtual "MoKey" Basado en Gestos para Personas con Movilidad Reducida: Capa del sistema,» Proyecto fin de grado, ETSI de Ingeniería de Telecomunicación, UPM, Madrid, Julio 2015.

[11] Microsoft, «https://msdn.microsoft.com/en-us/library/hh855347,» 2013. [En línea]. Available: https://msdn.microsoft.com/en-us/library/hh855347. [Último acceso: 14 Julio 2016].

[12] H. M. Hondori y M. Khademi, «A Review on Technical and Clinical Impact of Microsoft Kinect on Physical Therapy and Rehabilitation,» *Journal of Medical Engineering*, pp. 846514 -846530, 2014.

[13] D. Webster y O. Celik, «Systematic review of Kinect applications in elderly care and stroke rehabilitation,» *Journal of Neuroengineering and Rehabilitation*, vol. 11, 3 de Julio 2014.

[14] R. Madeira, L. Costa y O. Postolache, «PhysioMate - Pervasive physical rehabilitation based on NUI and gamification,» de *International Conference and Exposition on Electrical and Power Engineering (EPE)*, 16-18 de Octubre 2014.

[15] V. Group, «Virtual Rehab,» 2014. [En línea]. Available: http://www.virtualrehab.info/. [Último acceso: 13 Julio 2016].

[16] «KinectoTherapy,» 2012. [En línea]. Available: http://www.kinectotherapy.in/. [Último acceso: 12 Julio 2016].

[17] «Neumimic,» 2014. [En línea]. Available: http://startupcompete.co/startup-idea/business/neumimic/20828. [Último acceso: 13 Julio 2016].

[18] J. C. R. E. Centre, «KineLabs,» The Hong Kong Polytechnic University, 2012. [En línea]. Available: https://www.polyu.edu.hk/bme/kinelabs/. [Último acceso: 13 Julio 2016].

[19] Universidad Politécnica de Valencia, Compositor, *Kinect Umbrella.* [Grabación de sonido]. UPV Radiotelevisió. 2013.

[20] «OpenNI,» 2016. [En línea]. Available: http://openni.ru/openni-sdk/. [Último acceso: 13 Julio 2016].

[21] Delicode, «Ni-Mate,» 2016. [En línea]. Available: https://ni-mate.com/. [Último acceso: 13 Julio 2016].

[22] N. Steenbergen, F. Biermann y B. Walther-Franks, «BLOOP - Rapid Motion Capturing using,» 2011. [En línea]. Available: http://download.blender.org/documentation/bc2011/Bloop_Blender259.pdf. [Último acceso: 13 Julio 2016].

[23] Blender Foundation, «Blender 2.68 API,» 19 Julio 2013. [En línea]. Available: http://www.blender.org/api/blender_python_api_2_68_release/contents.html. [Último acceso: 14 Julio 2016].

[24] A. Hayter, «Kinect OSC,» 2013. [En línea]. Available: https://github.com/ahsquared/KinectOSC. [Último acceso: 13 Julio 2016].

[25] J. Zarco Torres, «Aplicación de un visor de realidad virtual a juegos serios para rehabilitación,» Proyecto fin de grado, ETSI de Ingeniería de Telecomunicación, UPM, Madrid, Julio 2016.

[26] «Kinect for Windows SDK v1.8,» Microsoft, 2016. [En línea]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=40278. [Último acceso: 13 Julio 2016].

[27] Microsoft, «Tracking Users with Kinect Skeletal Tracking,» 2013. [En línea]. Available: https://msdn.microsoft.com/en-us/library/jj131025.aspx. [Último acceso: 14 Julio 2016].

[28] C. A. Edo Solera, «Rotaciones en MatLab mediante matrices de rotación y cuaterniones,» [En línea]. Available: http://mural.uv.es/caraleso/robotizados/data/Memoria.pdf.

[29] M. Wright, «The Open Sound Control 1.0 Specification,» 26 Marzo 2002. [En línea]. Available: http://opensoundcontrol.org/spec-1_0. [Último acceso: 13 Julio 2016].

[30] «OSVR - Open Source Virtual Gaming,» Open Source Virtual Reality, 2016. [En línea]. Available: http://www.osvr.org/.

[31] «Android Debug Bridge,» Android Studio, 2016. [En línea]. Available: https://developer.android.com/studio/command-line/adb.html . [Último acceso: 13 Julio 2016].

[32] Android Developers, «Introduction to Android,» [En línea]. Available: https://developer.android.com/guide/index.html. [Último acceso: 14 Julio 2016].

[33] S. Miyamoto, T. Tezuka, E. Aonuma y otros, «The Legend of Zelda,» Nintendo, 1986-2016.

[34] J. Y. Peláez Castro, «Memoria de prácticas externas,» Escuela Técnica Superior de Ingenieria y Sistemas de Telecomunicación, UPM, Madrid, Julio 2016.

[35] P. Parra Iglesias, «Memoria de práctia externa,» Escuela Técnica Superior de Ingenieria y Sistemas de Telecomunicación, UPM, Madrid, Julio 2016.

Anexos

Anexo 1. Cómo instalar un *add-on* en Blender

Las funciones adicionales desarrolladas para Blender se han incluido en el programa como archivos de texto, que contienen un código que Blender puede leer e incorporar a su interfaz. Los scripts quedan instalados de forma duradera, quedando accesibles para su uso cada vez que se inicie el programa hasta que sean desinstalados, si se desea. Para importar el archivo se deben seguir los siguientes pasos:

1. Desde el menú *File*, o Archivo, abrir la ventana *User preferences*, Preferencias de Usuario. Se puede usar también la combinación *Ctrl+Alt+U*.

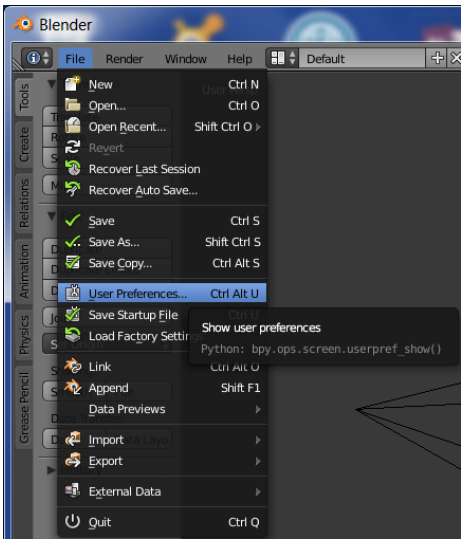

Figura 30 - Menú Archivo en Blender

2. En esta ventana, ir a la pestaña *Add-ons* y pulsar el botón *Install from File*, Instalar desde Archivo. Esto abrirá un navegador.

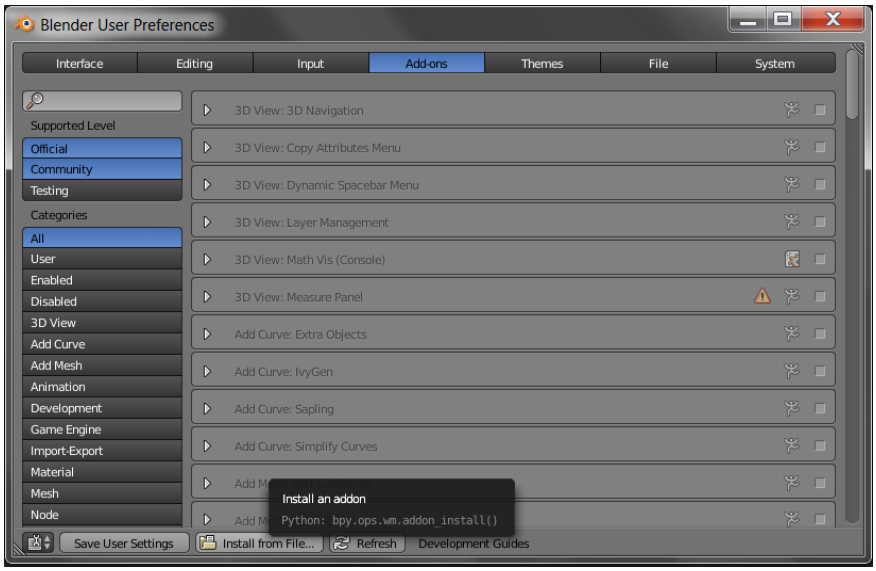

Figura 31 - Ventana de preferencias de usuario

3. En el navegador, elegir el archivo del *add-on*. Éste debe tener un formato de archivo de Python, *.py*, y cumplir con los requisitos especificados en [REF]. Hacer doble clic sobre él, o seleccionarlo y pulsar el botón *Install from File*.

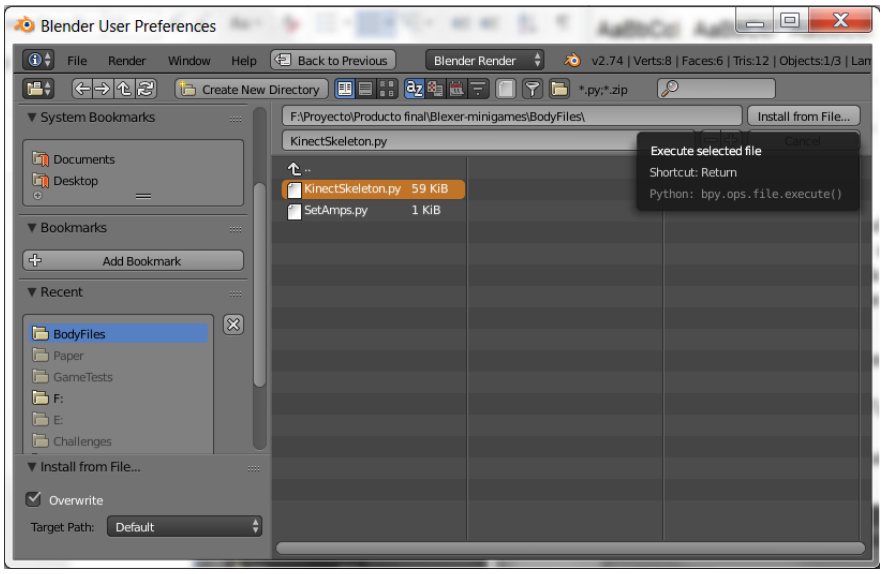

Figura 32 - Buscador de archivos

4. En la ventana de preferencias, de nuevo en la pestaña *Add-ons*, habrá aparecido un panel con los datos del fichero. Significa que el *add-on* ya está en la memoria de Blender. Hacer clic en la caja de su esquina superior derecha para habilitarlo y que se ejecute.

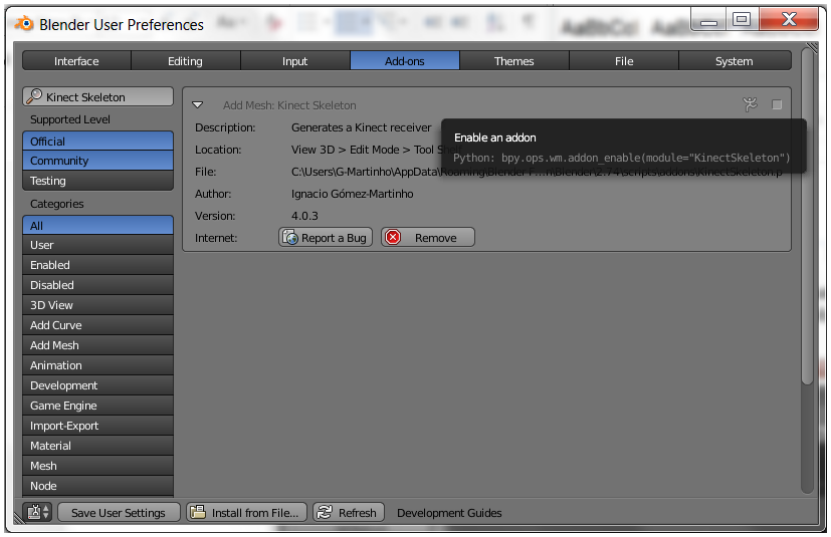

Figura 33 - Pestaña de add-ons

5. En esta misma ventana, conviene pulsar el botón *Save User Settings*, Guardar la Configuración de Usuario, para que los cambios queden registrados. Ya puede cerrarse la ventana de preferencias.

6. Si se quisiera deshabilitar el *add-on*, bastaría con pulsar de nuevo en la caja del punto 4 para deshabilitarlo. Si se pulsa el botón *Remove*, Eliminar, el *add-on* quedará borrado de la memoria de Blender.

Anexo 2. Cómo añadir una pestaña a Chiro

Se ha mencionado que el contenedor Chiro añade de forma automática a su interfaz todos los *middlewares* que encuentre referenciados en una lista. Este anexo explica cómo se debe crear un middleware compatible con Chiro y cómo editar la lista para incluirlo.

1. Crear una biblioteca

Por el diseño modular del Chiro, el código de los *middlewares* que se quieran añadir estará invariablemente en proyectos separados del código principal del contenedor. Chiro debe poder cargar el programa e incorporarlo a su entorno. La forma más fácil de lograr esto es que el proyecto del *middleware* no genere un archivo ejecutable, sino una biblioteca dinámica (*.dll*). Chiro podrá leerla y acceder libremente a todos los objetos que estén definidos en ella, principalmente la pestaña que contiene el *middleware*.

Para crear una biblioteca dinámica en Microsoft Visual Studio, hay que configurar un tipo de proyecto específico. Para crear un nuevo proyecto, hay que hacer clic en el menú *File*, Archivo, y elegir *New Project*, Nuevo Proyecto. También se puede usar la combinación *Ctrl+Shift+N*. Aparecerá una ventana donde se deben introducir todos los datos necesarios para crear el proyecto: el lenguaje en el que se codificará, su nombre, su ubicación y su tipo: de este último, se debe elegir *Class Library*, biblioteca de clases. Pulsando OK se creará el proyecto.

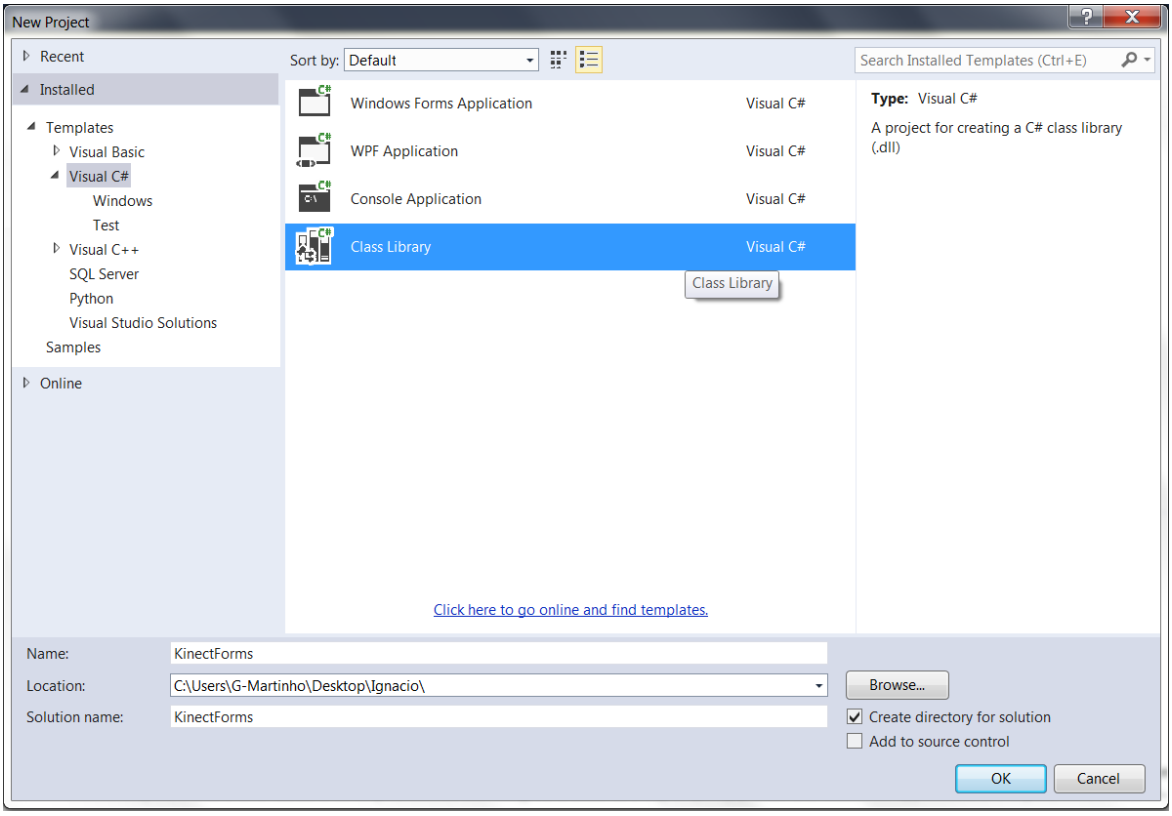

Figura 34 - Creación de un nuevo proyecto en Visual Studio

En este proyecto es donde se codificará el *middleware* que se quiere agregar a Chiro. Éste debe ser un objeto derivado de la clase Windows.Forms.TabPage, que es la clase de pestañas

que constituyen la interfaz de Chiro. Este objeto pertenecerá a un *namespace*, que es el conjunto de los objetos definidos en este proyecto. Puesto que Chiro no interactuará con la lógica interna del *middleware*, es muy importante que la ejecución de su constructor lo deje ya preparado para funcionar. Esto implica, entre otras cosas, que no puede necesitar que el programa principal le pase parámetros, puesto que éste no está diseñado para hacerlo.

Una vez se tenga el código completo y sin errores, el compilado generará el archivo *Path/NameSpace/obj/Debug/NameSpace.dll*, donde *Path* será la ubicación que se haya elegido para el proyecto y *NameSpace* será el nombre que se le haya dado.

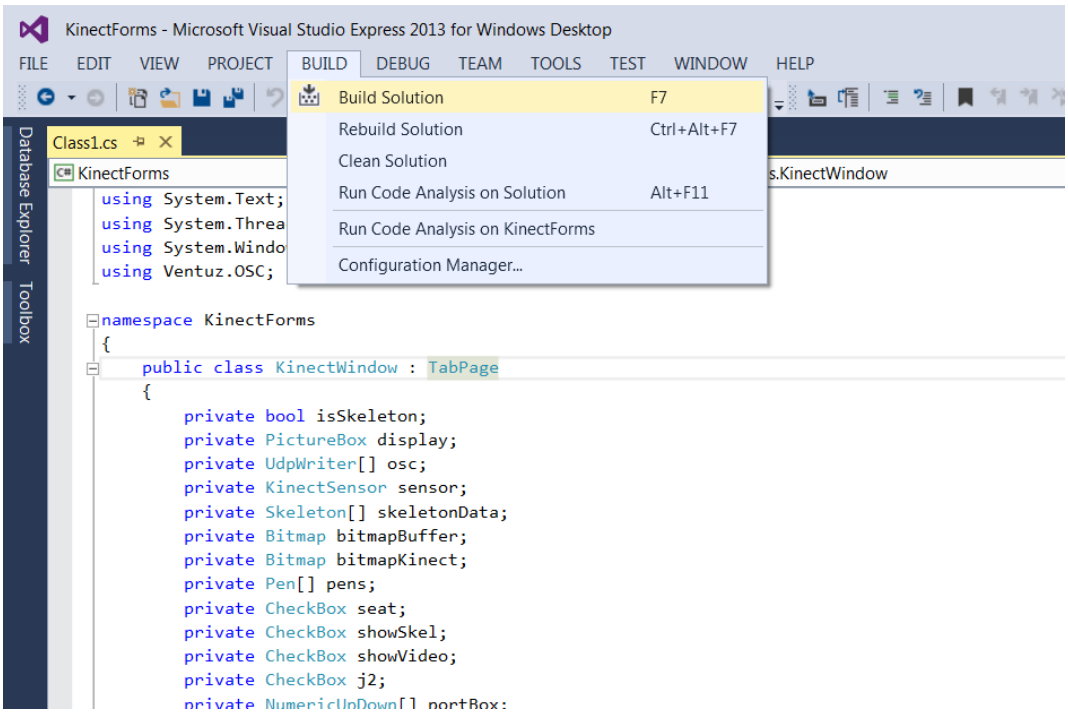

Figura 35 - Cabecera del código de la clase KinectWindow

2. Añadir una pestaña a la lista

Para que el contenedor incorpore a su interfaz la pestaña que se acaba de crear, ésta debe estar listada en el archivo que Chiro lee al iniciarse. Este archivo se encuentra en la ubicación *Chiro\bin\Debug\TabList.bin*. Es un archivo binario, por lo que no puede modificarse con un simple procesador de texto. Para editarlo se ha creado el proyecto de un programa ejecutable, el WindowRefMaker.

Este programa crea un conjunto de objetos tipo WindowRef: éste se ha definido en otro proyecto y se encuentra disponible en la librería dinámica *WindowReference.dll*, utilizada tanto por Chiro como por WindowRefMaker. Esta clase de objetos contiene una serie de datos referentes a las pestañas disponibles para Chiro. Los campos que deben ser definidos por el creador de un middleware son:

- El **título** de la pestaña.
- La ubicación de la **biblioteca** donde está definido el middleware. Esta ubicación debe expresarse en relación a *Chiro\bin\Debug*, pues ésa es la ubicación estándar para los archivos cargados por Chiro.
- El **objeto** que contiene el middleware. Debe expresarse de la forma *NameSpace.TabName*, donde *NameSpace* es el nombre de la biblioteca y *TabName* el de la pestaña.
- La **existencia** de un archivo ejecutable que permita instalar las bibliotecas necesarias para utilizar el *middleware*. En caso afirmativo, Chiro intentará ejecutar este archivo si ve que la carga de una pestaña falla.
- La ubicación de dicho **ejecutable**. En el caso de KinectWindow, éste sería el archivo que permita instalar el SDK de la Kinect.

La función del programa WindowRefMaker es crear una serie de objetos WindowRef con los valores que el desarrollador elija para estos campos. Actualmente no se ha implementado la función de introducir estos valores mediante consola, por lo que es necesario editar el código de WindowRefMaker, compilarlo, y ejecutar el programa. En la figura X se puede ver el fragmento decisivo de este código, con información para incluir dos pestañas.

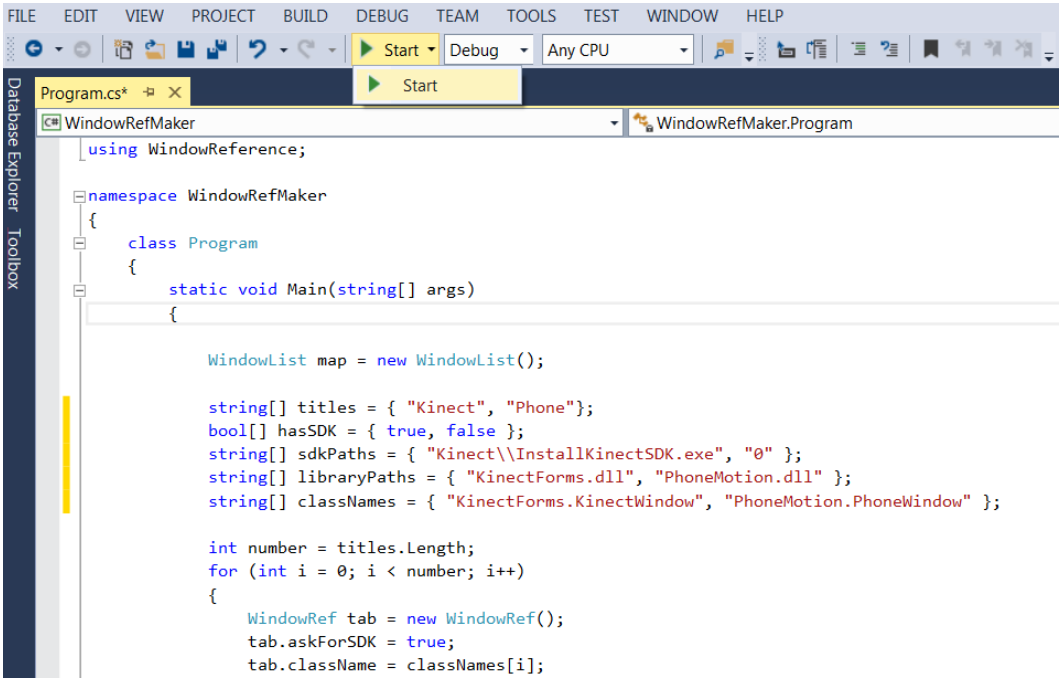

Figura 36 - Definición de los datos principales de la lista de objetos WindowReference

La ejecución del programa creará el archivo *WindowRefMaker\bin\Debug\TabList.bin*. Este archivo debe ser copiado manualmente y emplazado en la carpeta *Chiro\bin\Debug*. Al ejecutar el Chiro, éste procederá a leer los datos, cargar las pestañas, y si es necesario ejecutar los instaladores.

Anexo 3. Crear un perfil de usuario

En el apartado 3.2.4.2 se han descrito los cuatro scripts destinados a crear un archivo que contenga la información sobre un usuario necesaria para el funcionamiento de los amplificadores: su posición de reposo y el desplazamiento máximo de sus extremidades. Estas funciones deben ser llamadas sucesivamente a lo largo de la ejecución de un juego, de tal forma que las posiciones del jugador queden registradas.

Para las pruebas descritas en el apartado 4 se creó una aplicación de Blender muy simple, destinada a la creación de estos perfiles. Contenía únicamente un esqueleto de tipo combinado, con varias pequeñas esferas asociadas a las articulaciones. Su finalidad es permitir la monitorización visual del esqueleto receptor durante la ejecución, ya que las armaduras son invisibles. Este receptor tiene un campo de datos, *File*, en el que antes de la ejecución se debe escribir el nombre del fichero que se deberá crear. Es recomendable que tenga el nombre del usuario al que pertenece, y la extensión debe ser *.pkl*, un formato binario reconocible por Python.

Esta aplicación se ha estructurado con una cuenta atrás de tiempo, empezando en 90 segundos, que va ejecutando las funciones a medida que se alcanza el momento estimado oportuno.

En el segundo 75 se ejecuta la función de detección de la posición de reposo. En ese momento el usuario debe estar quieto. En el caso de los jugadores en sillas de ruedas, se recomienda que sus manos descansen sobre sus muslos, o sobre los reposabrazos de la silla. Esta pose queda guardada en la memoria temporal del programa.

Entre los segundos 70 y 5, con una frecuencia de treinta veces por segundo, se llama a la función de registro del desplazamiento. Durante este período de tiempo, el jugador debe mover sus brazos, piernas, torso y la cabeza todo lo que pueda en todas las direcciones posibles, de forma que sus desplazamientos máximos queden guardados en la memoria temporal. Durante las pruebas se recomendaba seguir la siguiente rutina:

- Comenzar en la posición de reposo.
- Levantar los brazos hacia arriba, extenderlos hacia delante, y luego hacia los lados.
- Levantar el pie derecho todo lo posible, y luego desplazarlo lo más posible hacia la derecha. Repetir con el pie izquierdo.
- Inclinar el cuerpo y la cabeza todo lo posible hacia delante, después hacia atrás, y luego hacia los lados.

En el segundo 0, se llama a la función que crea el archivo y guarda en él todos estos valores. Ya se puede cerrar la aplicación.

Con el perfil creado ya es posible aplicarlo a un minijuego con amplificación de movimientos. Basta con editar la propiedad File del esqueleto estático, y escribir el nombre del fichero con el perfil. Al iniciar el juego, los objetos amplificadores ajustarán su posición de reposo y sus factores de multiplicación automáticamente.