ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Diseño e implementación de un entorno virtual de ejercicios físicos, basados en captura de movimiento

AUTOR: César Luaces Vela

TITULACIÓN: Grado en Ingeniería Imagen y Sonido

TUTOR: Martina Eckert

DEPARTAMENTO: Teoría de la Señal y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Juan Manuel Meneses

VOCAL: Martina Eckert

SECRETARIO: Enrique Rendón

Fecha de lectura:

Calificación:

El Secretario,

Agradecimientos

A todos los profesores que he tenido a lo largo de estos últimos años por haberme sabido inculcar lo necesario para poder a llevar a buen término este proyecto. En especial a mi tutora Martina Eckert por haber confiado en mí por completo para su realización.

A mis amigos, que siempre han estado ahí contra viento y marea.

Y por supuesto, a mi familia, sin la cual, nunca podría haber llegado hasta aquí.

Resumen

Este proyecto tiene como objetivo principal, el diseño de un conjunto de herramientas software que permitan la transmisión de la información captada por la cámara de captura de movimiento Microsoft Kinect V2 al software de diseño de videojuegos Unity 3D, donde esta información debe ser procesada y aplicada a un modelo 3D.

Estas herramientas software son, por un lado, un middleware cuya función es la comunicación con el hardware Microsoft Kinect V2 y el posterior envío de los datos transmitidos por dicho hardware a Unity 3D, y por otro, un complemento para Unity 3D que permite la interpretación de los datos recibidos desde el middleware y su posterior utilización como controlador de los movimientos de un modelo 3D.

El motivo para llevar a cabo este objetivo es la implementación de estas herramientas software en el proyecto *BLEXER* (Blender Exergames), que, a lo largo de los últimos años, se ha ocupado de la creación de una serie de aplicaciones que permiten a usuarios con diversidad funcional motora, la consecución de los ejercicios de rehabilitación que habitualmente realizan, mediante ejercicios físicos que les son mostrados en forma de videojuegos.

Así, dado que Microsoft Kinect V2 permite conocer las posiciones espaciales y rotaciones asociadas a veinticinco articulaciones del cuerpo humano, es posible captar de manera muy eficiente el movimiento realizado por un usuario, de manera que, no solo es posible representar dicho movimiento en el modelo 3D utilizado, sino que también es posible variar la amplitud de este movimiento de manera que se amplifique su valor.

Mediante esta técnica de amplificación, se posibilita el acceso al videojuego a usuarios cuya diversidad motora física les inhabilite para la realización de ciertos ejercicios, de manera que, aunque el usuario no sea capaz de llevar a cabo por completo los movimientos necesarios para la consecución del ejercicio, el modelo 3D de la aplicación si los realice por completo.

Este proyecto incluye en su fase final una aplicación generada en Unity 3D donde se implementan todas estas características y que pretende ser una muestra de las posibilidades del diseño de videojuegos realizado desde este software.

Abstract

The main objective of this project is the design of a set of software tools that allow the transmission of the information captured by the motion capture camera Microsoft Kinect V2 to the game design software Unity 3D, where this information should be processed and applied to a 3D model.

These software tools are, on the one hand, a middleware which function is the communication (data transmission) between the hardware Microsoft Kinect V2 and Unity 3D, and on the other hand, a plug-in for Unity 3D that allows the interpretation of the data received from the middleware and their subsequent use as a controller of the movements of a 3D model.

The reason for the implementation of these software tools is their inclusion into the project *BLEXER* (Blender Exergames). In this proyect, over the past few years, a series of applications have been created that allow users with motor functional diversity, the performance of their usual rehabilitation exercises in support of video games.

As Microsoft Kinect V2 provides the spatial positions and rotations associated with twenty-five joints of the human body, it is possible to capture in a very efficient way the user's movements, such that it is possible to represent it in the 3D model, and, furthermore, the amplitude of this movements could be amplified in case of muscle weakness.

In this way, it is possible to provide the video game to users whose physical motor functionality does not allow the realization of certain exercises. And, even if the user is not capable of carrying out the movements necessary for the achievement of the exercise, the 3D model of the application will perform the full movements.

This project includes an application generated in Unity 3D where all these characteristics are implemented and which is intended to be a demonstration of the possibilities to design exercise video games with help of this software.

Índice de contenido

Índice de figuras

Lista de acrónimos

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

GAMMA: Grupo de Aplicaciones Multimedia y Acústica.

IK: Cinemática Inversa **(**Inverse Kinematic).

IR: Infrarrojos **(**InfraRed).

PFG: Proyecto de Fin de Grado.

SDK: Kit de desarrollo de aplicaciones **(**Software Development Kit).

UDP: Protocolo del nivel de transporte basado en el intercambio de datagramas (User Data Protocol).

USB: Conexión que posibilita el envío y la recepción de información (Universal Serial Bus).

Capítulo 1. Introducción

En la actualidad se han realizado importantes avances en materia de captura de movimiento y en su posterior procesado, apareciendo en el mercado gran cantidad de aplicaciones que utilizan estas tecnologías para su aplicación en distintos campos. Así, desde el grupo de investigación GAMMA [1] (Grupo de Aplicaciones Multimedia y Acústica) afincado dentro del CITSEM [2] (Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad), se ideó un proyecto que aunara estas tecnologías con un componente de intervención social, llamado proyecto *BLEXER*.

De esta manera, el proyecto en cuestión se basa en la creación de un entorno de juegos configurables que, mediante la captura de movimientos, permita a usuarios con diversidad funcional motora, la realización de una serie de ejercicios de rehabilitación en un entorno virtual creado específicamente para esta tarea. Así, se trata de alejar al jugador de la tarea habitual de rehabilitación acercándole lo máximo posible a la experiencia de participación de un videojuego al uso.

Esta herramienta software y los resultados que de ella se obtienen, deben suponer, por un lado, una mejora de la experiencia del usuario al realizar los ejercicios de rehabilitación, y por otro lado una herramienta de análisis del progreso del propio usuario por parte de su equipo médico, a través de los resultados obtenidos en los distintos ejercicios realizados. El acceso a estos resultados y la configuración inicial de cada ejercicio se realiza a través de una página WEB diseñada para este propósito en fases anteriores del proyecto, llamada *BLEXER-Med*.

Para la primera versión funcional del proyecto se creó una aplicación, llamada “Chiro”, que permite, por un lado, la comunicación con el hardware de captura de movimiento Microsoft Kinect V1 [3], y por otro, el envío de esta información a otra aplicación, llamada “Phiby´s Adventure”, en la que se reúnen varios de los ejercicios de rehabilitación previstos. Tras esta primera versión del proyecto, se decidió, basándose en el trabajo ya hecho en el PFG de Ignacio Gómez-Martinho González [4], generar una versión completamente nueva de las herramientas software diseñadas. El motivo de esta decisión fue, por un lado la adquisición de una nueva versión mejorada del hardware de captura de movimiento Microsoft Kinect, y por otro la necesidad de utilizar un nuevo entorno para el desarrollo del videojuego, dadas las muchas limitaciones que ofrecía el que se utilizaba en aquel momento.

Este PFG se enmarca dentro de este proyecto, enfocándose en el desarrollo de dicha herramienta software para la comunicación entre el nuevo hardware de captura de movimiento y el, también nuevo, entorno virtual de ejercicios, y en el desarrollo de la lógica de programación que rige los distintos ejercicios que el usuario realiza. El desarrollo de una primera versión de la comentada herramienta software o *middleware*, se realizó durante el

periodo de prácticas del alumno, usándose como base para el diseño e implementación de la versión final del *middleware* diseñado.

Así, dentro de este PFG se pueden discernir varios objetivos principales:

- Generar las herramientas software necesarias para conseguir una comunicación fluida y estable entre el hardware de captura de movimientos y el entorno de desarrollo de la aplicación.

- Procesar los datos recibidos desde el hardware en el entorno de desarrollo de la aplicación y transformarlos en movimientos.

- Simplificar el uso y acceso, tanto de la herramienta de comunicación con el hardware de captura de movimiento, como del propio software ya que, está ideado para que su uso final sea fuera del entorno académico, permitiendo que cada usuario pueda hacer uso de él sin un entorno específico de trabajo y sin necesidad de supervisión alguna.

- Desarrollar la lógica de programación asociada a los distintos ejercicios que se encuentran en el entorno virtual.

- Diseñar dicho virtual entorno y el *gameflow* del propio juego. Llamamos *gameflow* a la creación de situaciones de cierta dificultad para el jugador que se recompensan cuando son completadas, dándole una motivación para continuar el juego.

Capítulo 2. Marco tecnológico del proyecto

2.1. Estado del arte

En la actualidad, el mercado muestra un número considerable de aplicaciones cuyo objetivo principal es el de ser utilizadas como complemento para las distintas terapias médicas que cierto tipo de pacientes puedan necesitar. Dentro de estas aplicaciones destacan las que, mediante la utilización de algún tipo de hardware de captura de movimiento, tratan de ser un apoyo para la realización de los ejercicios de rehabilitación física que deben realizar los pacientes con cierto grado de diversidad funcional motora.

Dentro de este ámbito cabe destacar las aplicaciones que, por su nivel de implantación y número de pruebas exitosas realizadas, han demostrado su utilidad práctica como terapias de rehabilitación alternativa, recibiendo todas ellas una acogida muy positiva, tanto por los usuarios que las integraron en sus procesos de rehabilitación, como por el cuerpo médico que recomendó su uso.

- **AssessMS**

  Se trata de un software desarrollado en conjunto por Microsoft y la compañía médica *Novartis* [5] y está dirigido a pacientes que padezcan esclerosis múltiple amiotrófica (ELA). Su misión es, mediante la captura de movimientos con el hardware Microsoft Kinect, tratar de medir en estos pacientes, a lo largo de un periodo temporal, el grado de avance en los síntomas relacionados con la enfermedad, evaluando mejor su progreso y pudiendo acelerar el proceso de obtener los tratamientos adecuados para ellos (Figura 1).

  Actualmente, tras haber funcionado de forma exitosa con un grupo reducido de pacientes, se encuentra en fase de expansión, siendo su siguiente paso el de evaluar el software en la práctica con un grupo más amplio de usuarios.

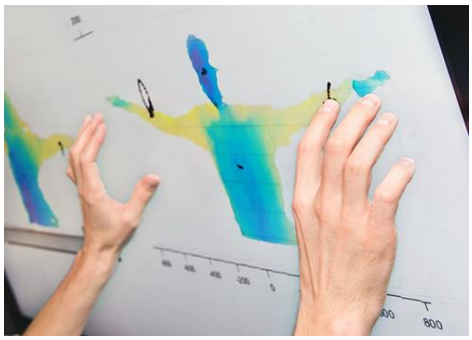


Figura 1. Ventana de muestra del software AssessMS

- **Human Motion**

Otro software, desarrollado también como complemento para los procesos de rehabilitación física, es *Human Motion*, diseñado por la empresa de desarrollo de aplicaciones *Precission Games* [6]. Human Motion es una aplicación que utiliza también el sensor Kinect para ayudar a los pacientes a través de su proceso de rehabilitación (Figura 2).

Se trata de una herramienta que ayuda a los profesionales a mejorar el estado físico de sus pacientes mediante fórmulas motivadoras. Además cuenta con un novedoso sistema de recopilación de información útil ayudando a una mejor toma de decisiones. Así, desde el punto de vista del paciente, con esta ayuda, el proceso de rehabilitación se vuelve menos tedioso y trata de que, de alguna manera, olvide la dolencia asociada a su caso. También se trata de una gran ayuda para el profesional, ya que permite mejorar la relación con sus pacientes, haciendo la terapia más dinámica y personal.

Figura 2. Ventana de muestra del software HumanMotion

- **VirtualRehab**

VirtualRehab [7] es un novedoso sistema de rehabilitación física basado en videojuegos que permite la monitorización y seguimiento de los pacientes desde cualquier lugar del mundo. Está desarrollado por la empresa *Virtual Ware* y se ha diseñado con el fin de que, los pacientes puedan realizar complejos programas de rehabilitación a través de terapias divertidas tanto en su centro de rehabilitación cómo en sus propios hogares (Figura 3).

El software está dirigido a profesionales en el ámbito de la salud, centros de rehabilitación, clínicas, hospitales, centros de día, asociaciones, etc, y se ha diseñado para el tratamiento de diferentes patologías como: Daño cerebral adquirido (Accidente Cerebrovascular (ictus), Traumatismo Craneoencefálico), Enfermedades neurodegenerativas (Esclerosis Múltiple, Párkinson, Esclerosis Lateral Amiotrófica, Alzhéimer), Enfermedades neuromusculares (Distrofias, Miopatías, Amiotrofias, Neuropatías), o Movilidad para tercera edad.

Además VirtualRehab es el primer producto de rehabilitación virtual que utiliza dispositivos como Microsoft Kinect que obtiene el marcado de conformidad con la directiva de la Comunidad Económica Europea (CE) como producto sanitario (PS), cumpliendo así todas las regulaciones necesarias en el mercado nacional e internacional.

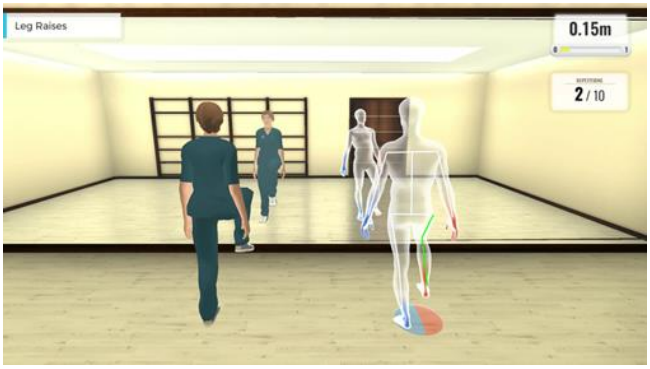


Figura 3. Ventana de muestra del software VirtualRehab

En esta misma línea de pensamiento se enmarca el proyecto *BLEXER*, desarrollado a lo largo de los últimos años por el grupo de trabajo GAMMA, que ha tenido como objetivo principal desde su creación, conseguir llevar a cabo la tarea de implementar un entorno virtual que permita a un usuario, con cierto nivel de limitación motora, la realización de una serie de ejercicios de rehabilitación, que, en cierto modo, se camuflan como eventos de juego de un videojuego.

Para ello, en las primeras fases de este proyecto se utilizó, para la captura de movimientos, el hardware de Microsoft Kinect V1 y, para la implementación del videojuego, el software de modelado 3D Blender [8]. Este software de modelado también incluye un motor que permite la ejecución de código ligado a los modelos 3D mediante scripts escritos en lenguaje Phyton. En la figura 4 se muestra la cámara Kinect V1 y una ventana de ejemplo del software Blender.

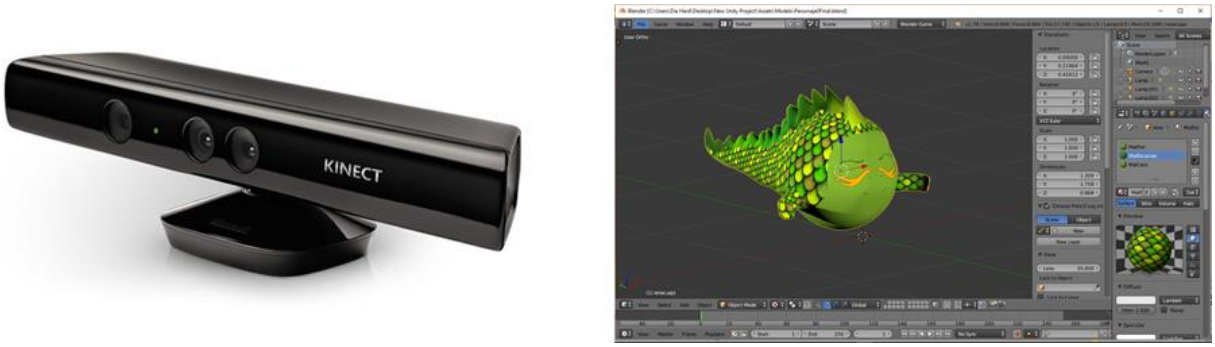


Figura 4. Hardware de captura de movimientos Microsoft Kinect V1 y software de modelado 3D Blender

En estas primeras fases se llega a implementar una serie de aplicaciones completamente funcionales en las que el usuario puede desplazarse con cierta libertad por un entorno, que va descubriendo al avanzar y, en el que se encuentran dispersos, a modo de objeto interactuables, los distintos ejercicios de rehabilitación.

Para dotar de mayor grado de jugabilidad al videojuego, también incorpora una mecánica de juego en la que cada ejercicio realizado genera cierto tipo de recursos, que son acumulados en la ficha del jugador y que permiten la fabricación de objetos que le habilitan para el acceso a nuevas zonas de juego.

El diagrama mostrado en la figura 5 [9] muestra cómo, además del desarrollo del sistema de comunicación entre Microsoft Kinect y el videojuego diseñado, también se implementa un sistema de comunicación y de almacenamiento de datos que permite al terapeuta de manera remota, tanto la modificación de las características de juego que definen cada ejercicio de rehabilitación, como el seguimiento de los progresos de cada paciente a través de los resultados almacenados en la base de datos asociada.

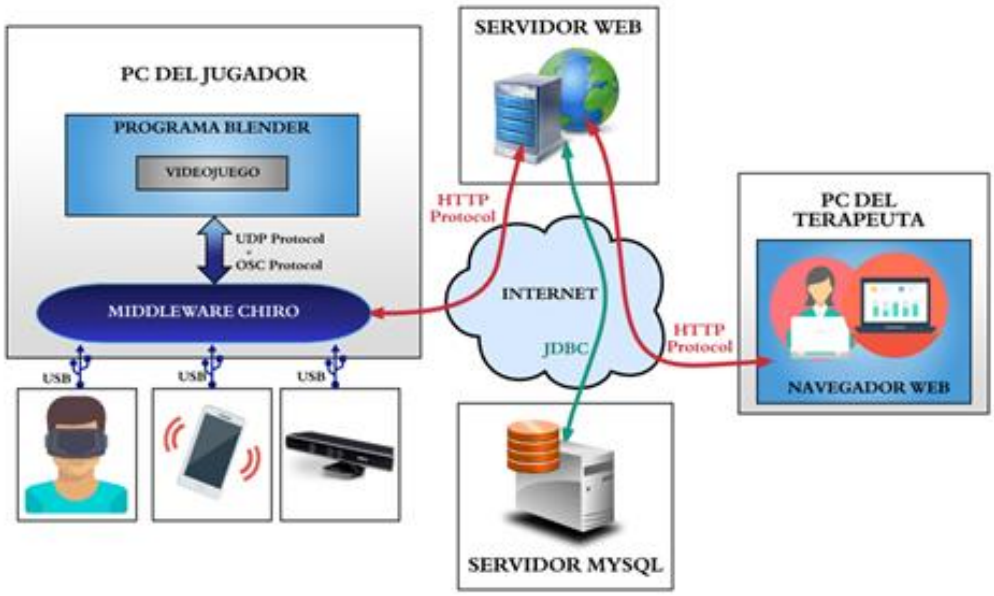


Figura 5 . Aplicaciones ligadas a las primeras fases del proyecto BLEXER

Para la gestión de la comunicación entre todas las aplicaciones que componen el sistema, dado que, por ejemplo, Blender no permite la comunicación de forma directa con Kinect, se diseña un *middleware* llamado "Chiro" que, no solo permite a Blender la recepción de datos proveniente de Kinect si no que, también permite la recepción de datos procedentes de otros dispositivos. De esta manera se trata de crear una herramienta modular que permita la utilización de distintos sensores en función de las necesidades de la aplicación demandante de información.

Tras estas primeras fases, a continuación se tuvo en cuenta, por un lado, la aparición de la versión dos del hardware Microsoft Kinect distribuido en conjunto con el sistema "Xbox One", que incluye toda una serie de mejoras y sustituyó rápidamente en el mercado a la anterior versión, y por otro, las limitaciones que, dado que es básicamente un software de modelado 3D, imponía el software de diseño Blender a la lógica de programación del videojuego.

Así, para esta nueva fase del proyecto *BLEXER*, en la que se enmarca este PFG, hay dos tareas principales a llevar a cabo. La primera de ellas, es el diseño de una nueva herramienta que permita la comunicación con el nuevo hardware a utilizar Microsoft Kinect V2, ya que, las órdenes de control y el formato de los datos transmitidos cambian por completo con respecto a la versión anterior. La segunda tarea es la implementación de los datos enviados por este hardware en el nuevo software de desarrollo Unity 3D [10], que dada su potencia, versatilidad y fácil acceso se ha impuesto como nueva herramienta de diseño.

2.2. *Microsoft Kinect V2*

Desde que en 2009 Microsoft lanza al mercado el capturador de movimientos Kinect, este hardware ha demostrado ser una de las mejores opciones, en relación calidad precio, cuando se desea realizar esta tarea. Así, tras el éxito de la primera versión, en 2013 Microsoft lanza al mercado una versión con mejoras en la mayor parte de sus características asociadas [11] y con nuevas funcionalidades incorporadas. La figura 6 muestra una imagen del hardware Kinect V2.

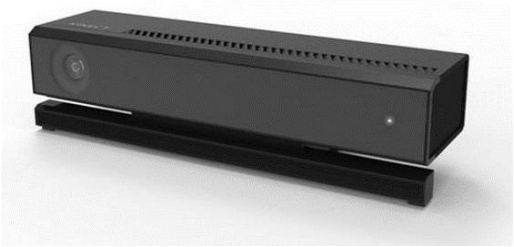


Figura 6. Hardware de captura de movimientos Microsoft Kinect V2

Características funcionales:

- Posee una resolución de 1920x1080 con un framerate de 30 fps y un formato de resolución de 16:9.
- Sus ángulos de visión son 70 grados en el eje horizontal y 60 grados en el eje vertical.
- La distancia mínima de uso es de 1.37 metros.
- Tiene un emisor de IR Activo, lo que le habilita para la visión nocturna.
- La latencia de transmisión es de 20 ms.
- Puede realizar la detección simultánea de 6 personas.
- Detecta 25 puntos del cuerpo simultáneamente.
- Puede realizar la detección de expresiones faciales, de los dedos y las muñecas o incluso de la forma del cuerpo detectado.

Debido a todas estas características, muy superiores a las de su versión anterior y a las de equipos de precio similar, y a la existencia de librerías de código abierto que permiten su libre uso a desarrolladores, este dispositivo es una opción muy interesante para el desarrollo de aplicaciones que impliquen la captura de movimientos de un usuario para su posterior interpretación.

2.3. *Unity 3D*

Unity 3D es un motor para videojuegos multiplataforma creado por Unity Technologies y está disponible como plataforma de desarrollo con interfaz gráfica. Es actualmente una de las plataformas de diseño con más éxito debido, a la potencia del motor gráfico asociado y al hecho de que, inicialmente, la licencia de uso es gratuita y solo se deben realizar pagos a la empresa asociada a este software cuando el diseñador obtenga beneficio del producto diseñado en él [12]. La figura 7 muestra una ventana de muestra del software.

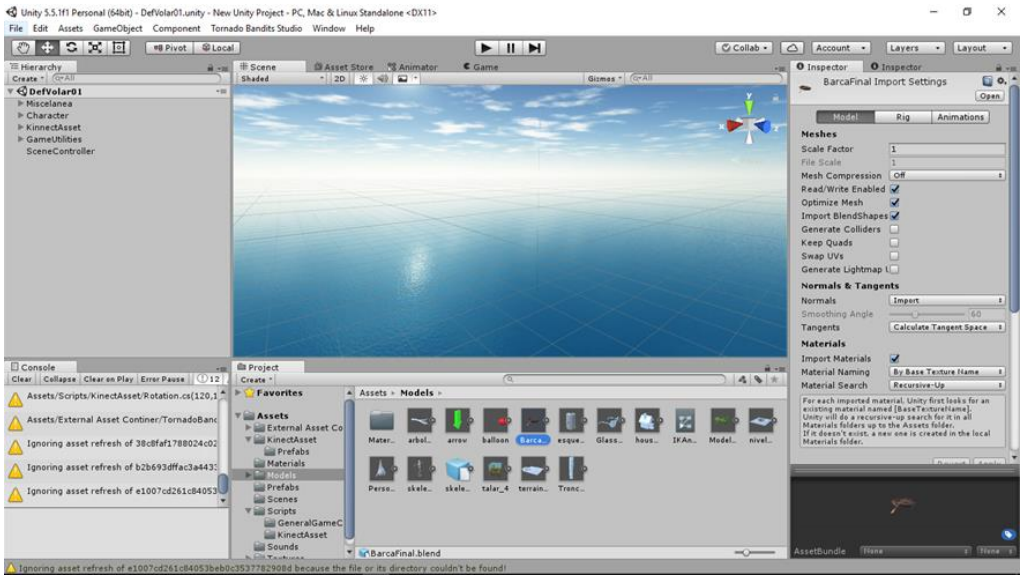


Figura 7. Software de diseño de videojuegos Unity 3D

Características funcionales:

- Permite el uso de una gran variedad de lenguajes de programación.
- También permite la importación de modelos y animaciones realizadas con otras aplicaciones de diseño 3D, como pueden ser Blender, Maya, 3ds Max, Modo, Cinema 4D, etc. Cuando esto se realiza, Unity 3D puede funcionar de forma simultánea con estas aplicaciones, de manera que actualizará los cambios que se realicen en el modelo 3D importado desde estas aplicaciones de diseño en todo el proyecto.
- Integra toda una serie de características de control desde las cuales se puede modificar de forma muy simple, todo lo relacionado con el comportamiento tanto físico como a nivel de iluminación de todos los objetos presentes en la escena y de ella misma.

Capítulo 3. Solución desarrollada

3.1. El Middleware

Dado que una de las bases de este proyecto es la comunicación entre el Hardware Microsoft Kinect V2 y el software de desarrollo Unity 3D, la primera decisión a tomar es como va a realizarse dicha comunicación. Así, han de barajarse las opciones de una comunicación directa entre los ejecutables generados en el entorno de desarrollo del videojuego y Kinect, o la creación de un *middleware* que actúe de intermediario entre ellos, tal y como sucedía en la versión anterior del proyecto.

En este caso, y pese a que Unity 3D sí posee el acceso librerías de código que permiten realizar la comunicación de forma directa con Kinect V2, lo que ahorraría el diseño de una herramienta aparte para esta función, se optó por la creación de un *middleware*. Esta opción se toma pensando en la versatilidad y compatibilidad de esta parte del proyecto con otras plataformas de desarrollo, además de la ya mencionada Unity 3D. De esta manera, si en un futuro se desease crear, mediante un software de desarrollo distinto, nuevas aplicaciones que necesiten de la comunicación con Kinect V2, la parte relacionada con dicha comunicación ya estaría implementada. Además, esto también puede ser aplicado al dispositivo utilizado para la captura del movimiento de manera que, si se desease utilizar otro distinto de Kinect V2 en un futuro, pudiera ser también fácilmente añadido al *middleware*.

Para el desarrollo de aplicaciones compatibles con este hardware de captura de movimiento Microsoft proporciona el kit de desarrollo oficial (SDK – *Software Development Kit*), versión 2.0 de Kinect [13]. En él, la mayor parte de los recursos que se encuentran son librerías para ser utilizadas en el lenguaje C#, de manera que este es el lenguaje en el que se implementa el *middleware*.

En la figura 8 se ilustra el funcionamiento completo de este sistema de transmisión.

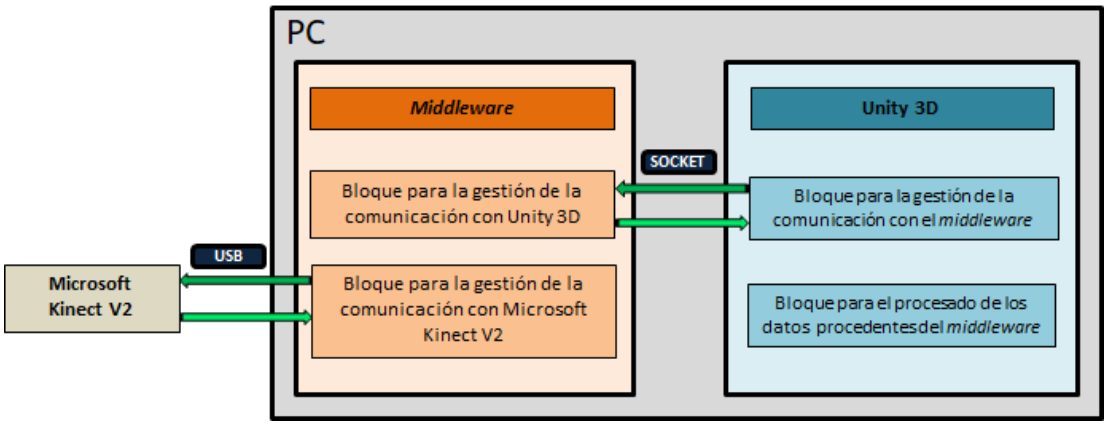


Figura 8. Diagrama de bloques general del sistema

De esta manera, en primera instancia, el *middleware* y el hardware Kinect V2 realizan una comunicación vía USB *(Universal Serial Bus*) mediante la cual el *middleware* almacena, de forma temporal, los datos asociados al movimiento de un individuo. Tras esto, dichos datos son procesados y empaquetados para su posterior envío a Unity 3D mediante un socket de transmisión interno. Este socket también permite a Unity 3D modificar las distintas opciones de configuración del *middleware* mediante mensajes de control.

3.1.1. Comunicación entre el middleware y Microsoft Kinect V2.

Tal y como se comenta en el apartado anterior, la función del *middleware* es guardar y actualizar la información sobre la pose de los jugadores, y transmitirla a la aplicación que haya solicitado dicha información. Así, el primer paso a dar para ello, es generar un canal de comunicación entre ambos sistemas que permita al *middleware* acceder a la información captada por Kinect V2.

Para realizar esta tarea el *middleware* utiliza los métodos asociados a la clase `KinectMiddleware.CS` **(Se lista, junto con todo el código implementado, en el Anexo III de este documento).** Esta clase, que ha sido generada íntegramente para este proyecto basándose en la documentación aportada por el SDK 2.0 de Kinect [14], se encarga de varias operaciones:

- **Gestionar la comunicación con Kinect V2, activando y desactivando el hardware**.
- **Acceder a la información de movimiento que suministra Kinect V2 vía USB**.
- **Procesar los datos recibidos desde Kinect para transmitir únicamente la información solicitada por la aplicación**.

A continuación se explica en detalle tanto el funcionamiento como la implementación de esta clase en el proyecto.

Se ha desarrollado mediante el entorno de desarrollo *Microsoft Visual Studio 2015*, que es también el software utilizado para el diseño de todos los scripts asociados a la aplicación.

Esta clase es la principal del *middleware* ya que se encarga de realizar la mayor parte de las tareas que este gestiona, con lo que, su explicación detallada es fundamental para poder comprender el funcionamiento específico del *middleware*.

La clase comienza comprobando si existe algún sistema Kinect conectado vía USB, y en caso de ser así, se conecta a él y le asigna una identificación única. Esta identificación se realiza porque existe la posibilidad de que una aplicación utilice más de una Kinect V2, con lo que, en ese caso, sería imprescindible poder distinguir la información proveniente de cada una de ellas.

Una vez detectado el hardware, se procede a generar las variables temporales donde se almacena la información recibida desde Kinect V2 y después a esperar a que llegue una solicitud de datos por parte de una aplicación que se conecte al *middleware*

La cámara Kinect V2 es un hardware de captura de movimiento que basa sus cálculos en la toma de imágenes de profundidad y en las escalas de color captadas, pero puede facilitar al usuario también, además de estas imágenes, otros datos de interés. Estos pueden ser, el sonido captado mediante un micrófono incorporado, la toma de imágenes con baja iluminación gracias a su sensor infrarrojo, o el reconocimiento de las emociones mostradas por el usuario.

Esta información se transmite a través de distintos *streams* que se activan cuando desde una aplicación se crea un lector de información de este tipo. Cuando uno de ellos está activo, lanza un evento indicando que un dato del tipo que es transmitido por él está disponible.

Así, el siguiente paso de la clase `KinectMiddleware.cs` es generar los lectores utilizados para comunicar con Kinect V2. Estos lectores, actúan como manejadores de los eventos lanzados desde el hardware, y dado que, en este caso las aplicaciones generadas en Unity 3D únicamente necesitan del acceso a datos relacionados con posiciones y rotaciones del cuerpo, solo es necesario generar un lector para los movimientos del cuerpo y otro para los de la cabeza. Aunque, como se ha visto, toda la información solicitada a Kinect V2 es del mismo tipo, estando relacionada únicamente con el movimiento captado, es necesario generar estos dos lectores debido a que Kinect V2 interpreta la cabeza como un elemento fuera del conjunto de articulaciones del cuerpo, y lo incluye dentro de la información relacionada con la cara. Por lo tanto es necesario activar un *stream* específico, que contiene toda la información asociada a la cara del usuario captado, para conocer la rotación realizada por la cabeza.

Una vez iniciada la transmisión, el hardware realiza una detección de los usuarios que se encuentran frente a la cámara. Es posible procesar el movimiento de hasta dos de los seis usuarios que Kinect V2 puede detectar, pero en el nuestro caso, dado que las aplicaciones previstas para implementar en Unity 3D no permiten la participación simultanea de más de un jugador, aunque el *middleware* transmite la información de todos los usuarios detectados, Unity 3D solo almacena y trata la información relacionada con uno de ellos.

La información asociada al movimiento del cuerpo captado es procesada por Kinect V2 y asociada un elemento tipo *body* que, de forma periódica, es transmitido en forma de evento a través del *stream* asociado a este tipo de datos.

Al igual que, a parte de la información relacionada con el movimiento del usuario, Kinect V2 proporciona una gran cantidad de datos acerca de la imagen captada, el elemento tipo *body* generado también contiene mucha más información de interés además de la relacionada con las posiciones y rotaciones de cada articulación. El estado de detección del usuario, es decir, si existe algún error en la detección del esqueleto por parte de Kinect V2, la altura a la que se encuentra el plano del suelo o el estado abierto o cerrado de las manos, son otros datos de interés asociados a este elemento que son tenidos en cuenta a lo largo de la ejecución de la aplicación.

Aunque esta información es útil para la aplicación, los datos realmente importantes para conseguir imitar los movimientos del usuario captados por Kinect V2, son los asociados a las posiciones espaciales y rotaciones de cada una de las articulaciones que son detectadas.

La figura 9 muestra todas las articulaciones detectadas por Kinect V2 y las distintas relaciones de dependencia entre ellas [15].

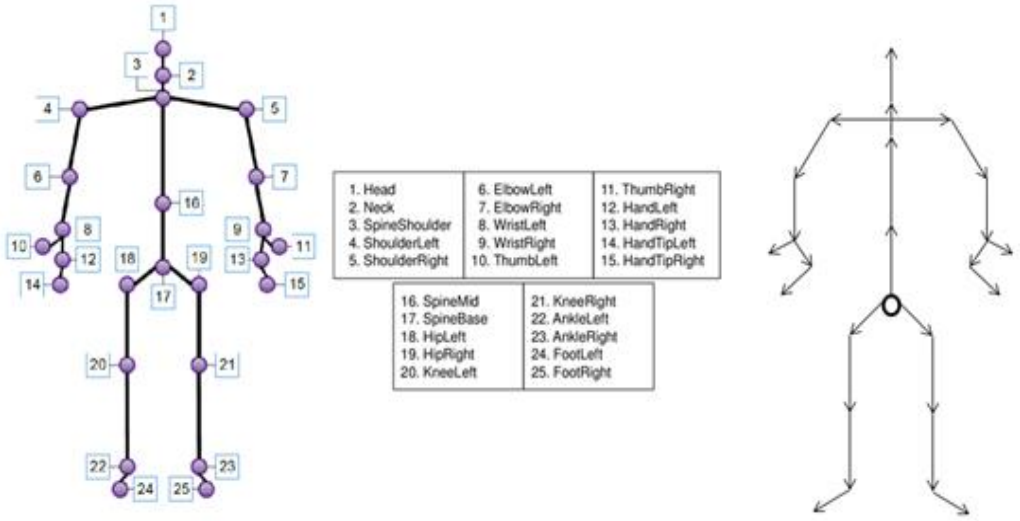


Figura 9. Nombre y posición de las articulaciones y relación de jerarquías entre las ellas.

Kinect V2 es capaz de detectar un total de 25 articulaciones, y generar un elemento tipo *body*, que contiene toda la información asociada al movimiento captado, con una tasa de envío de 30 por segundo, es decir, un paquete cada 33 ms aproximadamente.

Otro dato a tener en cuenta es que todas y cada una de estas articulaciones están emparentadas entre ellas siguiendo un orden jerárquico concreto. De esta manera, cuando una articulación rota, el movimiento se aplica a todas las articulaciones que dicha articulación tenga emparentadas, cambiando la posición que estas tienen en el espacio.

Teniendo en cuenta esto, el eje central del cuerpo del que dependen el resto de articulaciones es el punto de la cadera (17) llamado "*SpineBase*". Este punto base es el origen de una jerarquía de huesos, donde, dirigiéndose hacia las extremidades, cada hueso conectado depende del anterior y el ultimo es de menor orden jerárquico.

En la figura 10 se ilustra un ejemplo de movimiento que afecta a un sistema jerárquico de articulaciones, en el cual se aplica una rotación de 45 grados sobre uno de los ejes de rotación del hombro derecho de un esqueleto de prueba. Observando ambas imágenes se puede comprobar cómo, tras la rotación aplicada al hombro derecho, todas las articulaciones a él emparentadas, cambian su posición en el espacio, pero mantienen las posiciones y rotaciones relativas que tenían con respecto a la articulación que ha realizado el giro.

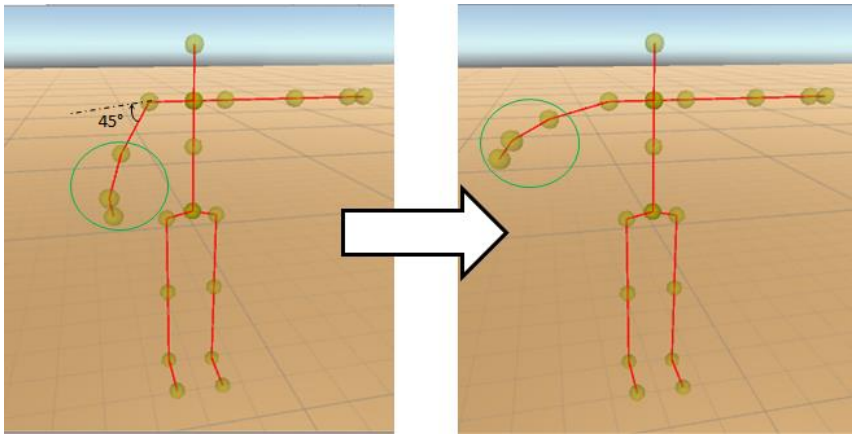


Figura 10. Ejemplo de rotación en un sistema con emparentamiento de articulaciones.

Teniendo en cuenta lo mostrado en este ejemplo, es posible definir dos sistemas de referencia de ejes cartesianos a los que pueden estar referidos los datos obtenidos en la comunicación con Kinect V2, el sistema global y el local:

- **Sistema Global:** El origen de coordenadas se encuentra en un punto concreto del espacio que es compartido por todas las articulaciones y no varía. Así, con cada movimiento realizado los valores de posición y rotación de cada articulación varían.

- **Sistema Local:** El origen de coordenadas es distinto para cada articulación y se encuentra asociado a la siguiente articulación que cada una de ellas tenga asociada por encima en el orden jerárquico, de manera que varía en cada movimiento realizado. De esta manera se consigue que, los valores de posición y rotación de una articulación varíen solamente cuando esta realiza un movimiento, y no cuando lo hace otra articulación con un orden jerárquico superior a ella.

Así, tanto los valores asociados a la posición espacial de cada articulación, expresados en forma de vector tridimensional, como los relacionados con su rotación, expresados en forma de cuaternión, son distintos en función del sistema de referencia utilizado (Figura 11).

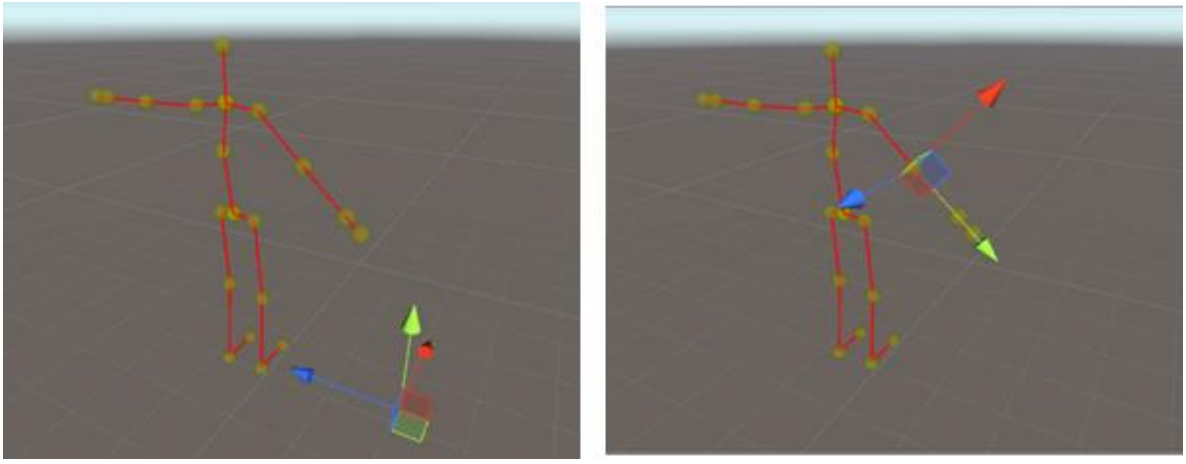


Figura 11. Sistemas de referencia de ejes cartesianos tipo global y tipo local

En la figura 11 se puede comprobar un ejemplo de lo comentado anteriormente, en el que, en un caso se muestra un sistema de referencia tipo global, con el origen de coordenadas situado a los pies del modelo, y en otro caso un sistema de referencia tipo local con el codo derecho como origen de coordenadas.

A continuación se definen estos dos tipos de valores, que, como se ha comentado, son de vital importancia a la hora de reproducir los movimientos captados por Kinect V2.

- **Valores asociados a la posición:** Tal y como se ha comentado en puntos anteriores de este documento, el sistema Kinect V2 proporciona de forma periódica un elemento tipo *body* con la información asociada al movimiento del esqueleto detectado por el hardware. Los datos asociados a la posición espacial de cada articulación se facilitan al usuario en un formato que el SDK de Kinect V2 denomina *CameraSpacePoint*. Tal y como indica su nombre, esta variable se relaciona con una posición espacial, y está definida como una estructura con tres valores asociados para almacenar las coordenadas X, Y y Z respectivamente. También, como se visualiza en la figura 12 tiene las siguientes características:

    - El **origen de coordenadas** se encuentra en el centro del sensor infrarrojo de Kinect V2.

    - La **coordenada X** crece hacia la parte izquierda del sensor.

    - La **coordenada Y** crece hacia la parte superior del sensor, teniendo en cuenta que esta dirección se basará en su inclinación.

    - La **coordenada Z** crece hacia la parte exterior del sensor en la dirección que se enfrenta a él.

Figura 12.– Posición y direcciones de crecimiento de las varibles X Y Z asociadas al origen de coordenadas de un CameraSpacePoint

De esta manera, con cada evento de movimiento, se obtiene un punto en el espacio para cada articulación almacenada en el elemento tipo *body*, de las que, si se unen con líneas rectas siguiendo la jerarquía indicada anteriormente en la figura 9, se obtiene el esqueleto del cuerpo detectado.

Este proceso se muestra en la aplicación de escritorio contenida en el SDK V2.0 de Kinect *KinectStudio* (Figura 13*). N*o se ha utilizado en el desarrollo del proyecto, es muy útil para visualizar la detección de un usuario por parte del sistema Kinect V2. También se usará para la realización de pruebas de funcionamiento en apartados futuros de este documento.

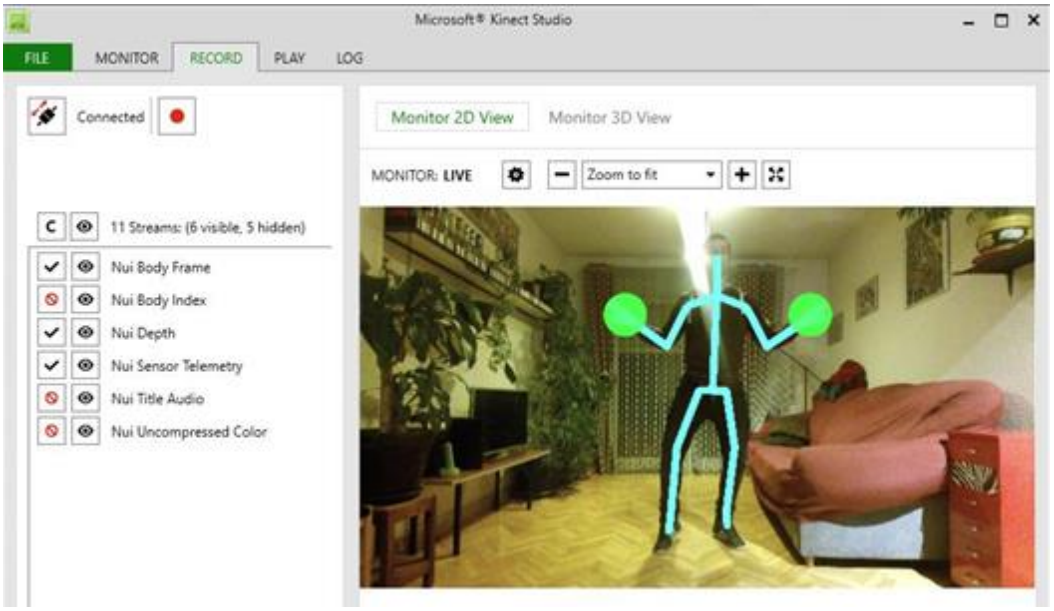


Figura 13. Ventana principal de la aplicación "Microsoft Kinect Studio"

- **Valores asociados a la rotación:** Al igual que el elemento tipo *body* transmitido contiene un tipo de variable específica para hacer referencia a la posición espacial de una articulación, también contiene una que hace referencia a su rotación. Este tipo de variable es denominada en el SDK V2.0 de Kinect como *jointOrientation*, y está definida como una estructura con cuatro valores asociados para almacenar las coordenadas asociadas al cuaternión calculado.

Un cuaternión [16] se define como una extensión de los números complejos creada con el fin de modelizar rotaciones en el espacio tridimensional, del mismo modo que con los números complejos se pueden modelizar rotaciones en dos dimensiones.

Su forma general es la siguiente:

$$q = s + xi + yj + zk \qquad s, x, y, z \in R$$

Donde *i*, *j* y *k* son números complejos que cumplen las siguientes propiedades:

$$i^2 = j^2 = k^2 = ijk = -1$$

$$ij = k \quad jk = i \quad ki = j$$

$$ji = -k \quad kj = -i \quad ik = -j$$

En la figura 14 se visualizan de forma gráfica estas propiedades descritas. En ella se puede comprobar cómo, dado que el producto escalar de dos de los distintos números complejos que componen un cuaternión da como resultado siempre el restante con uno u otro signo, es posible, modelizar los tres planos que componen el espacio tridimensional, permitiendo realizar la rotación de un vector en sus tres dimensiones, mediante los tres números complejos que definen respectivamente la rotación asociada a cada uno de los tres planos descritos.

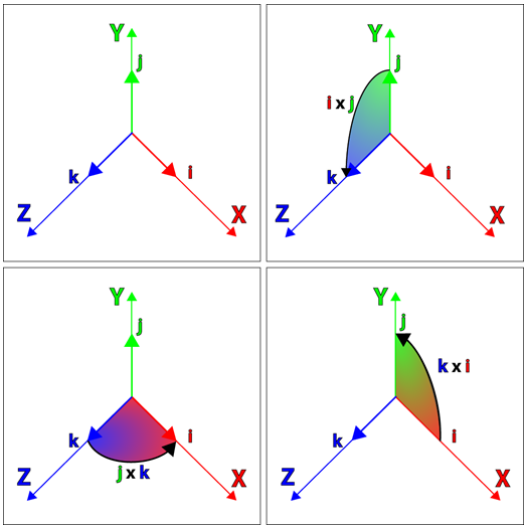


Figura 14. Representación gráfica de las propiedades de los cuaterniones[16]

Pese a que el papel de los cuaterniones fue poco significativo en el mundo de las matemáticas, ya que fueron sustituidos por el producto escalar y vectorial poco después de su creación, en la programación en 3D se ha encontrado recientemente su nicho de utilización. En ella han demostrado ser más eficaces que el producto escalar y vectorial debido a que, fundamentalmente, frente a los nueve números de una matriz que utilice un sistema de representación basado en vectores tridimensionales, permiten definir una rotación en el espacio empleando para ello tan solo cuatro números, consiguiendo con ello una reducción considerable en el tiempo de procesado.

Además es mucho más fácil para el procesador realizar una interpolación lineal esférica entre dos rotaciones que entre dos vectores, de manera que, los cálculos realizados por este con cuaterniones para dar fluidez al movimiento, requieren menos operaciones y son menos costosos en tiempo de procesado.

Así, debido a estas características, se toma la decisión de utilizar los cuaterniones, en lugar de las posiciones espaciales, como referencia para informar al software Unity 3D de los movimientos realizados por el usuario y detectados por Kinect V2.

3.1.2. Comunicación entre el middleware y Unity 3D

Una vez el *middleware* ha obtenido los datos asociados al movimiento que Kinect V2 envía, la siguiente fase del proceso de comunicación es transmitir estos datos almacenados al software donde van a ser procesados, en este caso Unity 3D.

Para decidir el sistema de transmisión a utilizar para esta comunicación, hay que tener en cuenta las necesidades de la aplicación que posteriormente se desarrollará en Unity 3D. La función principal de esta aplicación es trasladar en tiempo real los movimientos detectados por Kinect V2 a un modelo 3D generado en Unity 3D, haciendo que este se mueva de manera idéntica al usuario captado.

Para la realización de esta tarea, el sistema de transmisión debe elegirse teniendo en cuenta, por un lado, que la velocidad de transmisión de Kinect V2 es de un paquete cada 33 ms aproximadamente, y que, por otro, la información de cada nuevo paquete no está relacionada con el anterior, es decir, que si uno de ellos se pierde o si su información almacenada es errónea, Unity 3D puede completar el movimiento con el siguiente paquete correcto recibido. Así, en el sistema de transmisión a utilizar debe primar la velocidad de transmisión y no el control de la correcta recepción de cada paquete transmitido. Teniendo en cuenta esto, el protocolo de transmisión que mejor se adapta a estas necesidades es el UDP (User Datagram Protocol) [17], de manera que, es el que utiliza el *middleware* para llevar a cabo esta tarea. Este proceso se realiza a través de las clases `UDPSend.cs` y `UDPReceive.cs`.

Para llevar a cabo el proceso de comunicación anteriormente descrito, es necesario que el *middleware* genere varios hilos secundarios que se encarguen de su gestión. Cada uno de estos hilos secundarios tiene una función distinta, de manera que, mientras uno de ellos se encarga de comunicar con Kinect V2 y añadir los datos suministrados por el hardware a una cola de envíos pendientes compartida por todos los hilos, los otros dos hilos se encargan de la recepción y el envío de los paquetes UDP. Así, el funcionamiento en conjunto del *middleware*, se basa en un hilo principal o manejador, que, por un lado, crea los hilos secundarios que se encargan de la comunicación con Unity 3D y con Kinect V2, y que, por otro, analiza y procesa los datos que estos hilos gestionan. En la figura 15 se muestra de forma gráfica la progresión de estos hilos a lo largo del proceso de ejecución de la aplicación.

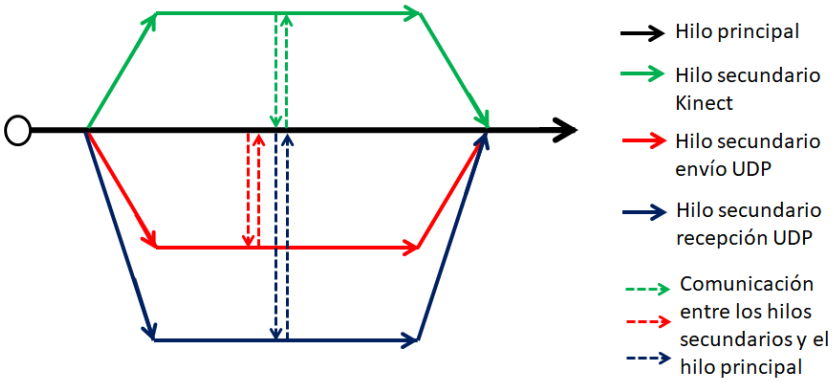


Figura 15. Progresion de los hilos de la aplicación a lo largo del proceso de ejecucion

3.1.2.1. Sistema de codificación en la transmisión

Una vez generado un canal de comunicación bidireccional estable entre el *middleware* y Unity 3D, es necesario decidir la codificación que se debe aplicar a la información transmitida por él, de manera que, esta pueda ser entendida de forma correcta por ambas aplicaciones. Para ello es necesario generar un sistema de codificación propio que permita, tanto la transmisión de los datos que Kinect V2 suministra, como el intercambio de información que modifica el comportamiento de las aplicaciones en tiempo de ejecución.

Este sistema de codificación, que utiliza como referencia un protocolo de características similares llamado *KinectOSC* [18], se basa en empaquetar la información de control del mensaje en cadenas de texto de tamaños concretos y, a continuación, los datos asociados al mensaje como otra cadena de texto de tamaño no definido. La figura 16 muestra de forma gráfica este formato de construcción.

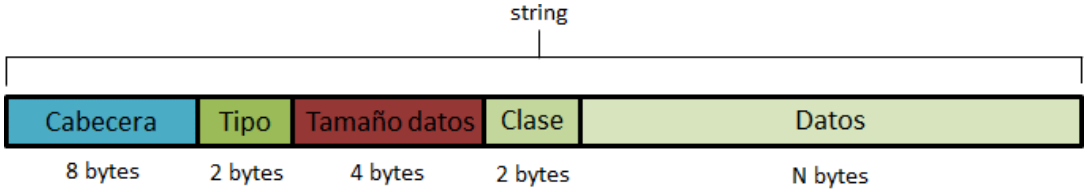


Figura 16. Formato de construccion de un paquete para la transmision de informacion entre el middleware y Unity 3D.

- **Cabecera:** Cadena de caracteres con la etiqueta *#K2UMUDP*, que distingue de forma inequívoca los mensajes pertenecientes al *middleware*. Esta etiqueta ha sido elegida durante la creación del protocolo y es debida al nombre puesto al *middleware: "K2UM"* (Kinect to Unity Middleware).

- **Tipo:** A lo largo del proceso de comunicación que realizan el *middleware* y Unity 3D, hay dos tipos de mensajes que el sistema puede transmitir. Unos relacionados con **información de control** del funcionamiento, tanto de Unity 3D como del *middleware*, y otros que contienen los **datos de movimiento** captados por Kinect V2. Mediante esta etiqueta se informa al receptor del mensaje del tipo de información contenida en él, en concreto, toma el valor *CC* para datos de control y *DD* para datos de movimiento.

- **Tamaño de los datos:** Tamaño en bytes de los datos transmitidos en el mensaje. Se utiliza como información de control de errores en recepción comprobando si el tamaño del *string* de datos coincide con la información almacenada en este campo.

- **Clase:** Según el tipo de información transmitida, es decir control o datos, este campo indica, o bien qué mensaje de control ha sido enviado, o bien qué clase de datos se envían en el mensaje. Más abajo, se especificará que siglas se usan en cada caso.

- **Datos**: Los datos asociados al mensaje, relacionados con mensajes de control que modifican el funcionamiento del sistema en tiempo de ejecución, o con el movimiento a aplicar al modelo 3D presente en Unity 3D.

3.1.2.2. Tipos de mensajes utilizados en la transmisión

En este apartado se detalla el contenido de los diferentes mensajes por el sistema, los relacionados con el control del *middleware* y de Unity 3D y los relacionados con los datos de movimiento transmitidos.

Se explica, tanto la función asociada, como el contenido de los distintos campos, mostrando también en forma gráfica la estructura real de la cadena de texto de datos asociada al mensaje enviado.

- **Mensajes de datos**

  El campo "Tipo" contiene la cadena de texto *DD* en y el campo "Datos" la información asociada al movimiento. Siempre son enviados desde el *middleware* a Unity 3D y, además de los datos asociados a cada articulación, también contienen el identificador que Kinect V2 asocia al usuario detectado. De esta manera, en caso de que hubiese más de un usuario detectado por Kinect V2, Unity 3D puede distinguir entre la información recibida de ambos. Estos mensajes pueden ser de dos tipos:

  - **Datos de posición (Figura 17):** Contienen la cadena de texto *PP* en el campo clase. Los datos transmitidos contienen información relativa a las tres coordenadas espaciales de cada articulación, precedida del nombre de la misma. Para indicar la finalización de la cadena de texto que contiene la información se utiliza la sucesión de caracteres *END*.

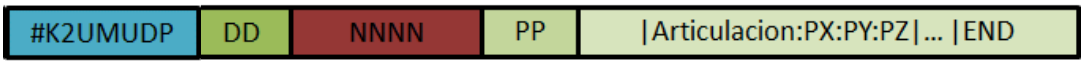


Figura 17. Estructura de un mensaje de datos de clase posición

  - **Datos de rotación (Figura 18):** El campo "Clase" contiene la cadena de texto *QQ*. De forma análoga a la clase anterior, los datos transmitidos contienen información relativa a las cuatro variables asociadas a un cuaternión, precedidas del nombre de la articulación a la que se aplica la rotación. Para indicar la finalización de la cadena de texto que contiene la información se utiliza la sucesión de caracteres *END*.

Si únicamente se ha solicitado el envío de mensajes de esta clase, el *middleware* incluye también una cadena de texto con el nombre *PBSpineBase* con la posición espacial de la articulación *SpineBase*, de manera que, siempre se pueda conocer la posición en el espacio del eje jerárquico del esqueleto.

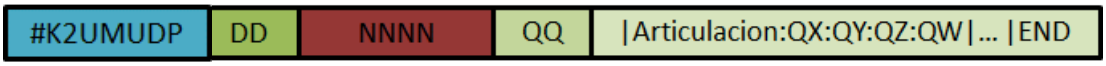


Figura 18. Estructura de un mensaje de datos de clase rotación

- **Mensajes de control**

Estos mensajes se identifican mediante la cadena de texto *CC* en el campo "Tipo" y contienen información asociada al funcionamiento del *middleware* y Unity 3D. Pueden ser enviados tanto desde el *middleware* a Unity 3D como desde Unity 3D al *middleware* y engloban tanto mensajes únicamente de control como mensajes de error asociados a la recepción de otros mensajes de control previos o problemas en la conexión con Kinect V2.

- **Información de articulaciones activas (Figura 19):** El campo "Clase" contiene la cadena de texto *AJ*. Se envía desde la Unity 3D al *middleware* la primera vez que se conectan y contiene una cadena de caracteres con las articulaciones activas en el modelo 3D utilizado, de manera que, solamente se envíe la posición o rotación de estas a la aplicación.

  Mediante esta técnica se consigue reducir tanto el tamaño de los mensajes enviados a la aplicación como el tiempo que tarda esta en procesar cada paquete recibido, ya que la información contenida es menor.

  Hay que tener en cuenta que, dado que Kinect V2 únicamente envía datos con un sistema de referencia de tipo global, comentado en el punto 3.1.1 de este documento, cuando se envía una articulación activa es imprescindible enviar también en este mensaje, todas las articulaciones asociadas a ella con un orden jerárquico superior en la cadena mostrada en la figura 9. En caso contrario, el movimiento aplicado a la articulación solicitada no sería el correcto ya que, en un sistema de referencia tipo global la posición y rotación de todas las articulaciones cambia con cada movimiento. De esta manera, la posición y rotación de una articulación depende también de las posiciones y rotaciones de todas las articulaciones asociadas a ella con un orden jerárquico superior.

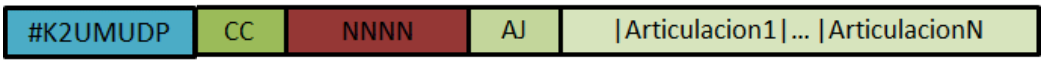


Figura 19. Estructura de un mensaje de control de clase articulaciones activas

- **Solicitud de datos de movimiento (Figura 20):** El campo "Clase" contiene la cadena de texto *SP*. Se envía desde la aplicación al *middleware* para solicitar el envío de un mensaje de tipo "Datos" con la última información de movimiento captada.

  Este envío se realiza cada vez que la aplicación ha procesado un dato de movimiento recibido o han pasado cinco ciclos desde el envío del último paquete de este tipo sin recibir contestación. En ese caso, la aplicación trata de mandar cinco veces este mensaje y en caso de no recibir el mensaje con los datos de movimiento informa al usuario de una desconexión con el *middleware*.

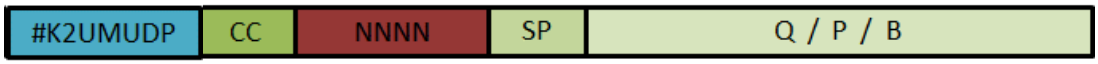
A su vez en el campo de datos este mensaje contiene un carácter que indica al sistema qué clase de mensaje de datos enviar:

- **Q:** Mensaje de clase "Datos de rotación".
- **P**: Mensaje de clase "Datos de posición".
- **B:** Envío de ambas clases de mensajes de datos. En este caso el *middleware* envía un mensaje por cada clase de datos solicitada, con lo que se duplica el número de paquetes UDP que envía.

Así, con este mensaje de control se trata de, además de, informar en todo momento al *middleware* de la clase de datos solicitada desde la Unity 3D, generar un sistema de comunicación que dote de cierta sincronía a la transmisión realizada por el socket UDP.

Pese a que la frecuencia nominal, es decir el número de operaciones que realiza por segundo, de Unity 3D es muy superior a la de refresco de Kinect V2, existe la posibilidad de que, por situaciones de saturación del procesador, esto cambie, y como consecuencia, caiga de forma drástica dicha frecuencia nominal. Este hecho podría llevar a la desincronización entre los datos enviados por Kinect V2 y los procesados por Unity 3D, lo que ocasionaría una reproducción de movimientos incorrecta en el modelo 3D.

Con este mensaje, se soluciona esta posible eventualidad al solicitar siempre la última de información movimiento captada por Kinect V2.



F

Figura 20. Estructura de un mensaje de control de clase solicitud de datos

- **Desconexión desde la aplicación (Figura 21):** El campo clase contiene la cadena de texto *PM*. Se envía desde la aplicación al *middleware* para indicar que ha finalizado su ejecución. No contiene información en el campo de datos.

Cuando el *middleware* recibe este mensaje indica al usuario que no hay ninguna aplicación conectada y se prepara para recibir el nuevo mensaje de control con la información de las articulaciones activas para el próximo envío. Hasta ese momento el *middleware* mantiene suspendido el envío de mensajes datos.

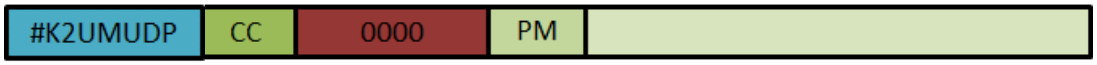


Figura 21. Estructura de un mensaje de control de clase desconexion desde la aplicación

- **Nombre de usuario (Figura 22):** El campo "Clase" contiene la cadena de texto *UN*. Este tipo de mensajes se envía desde Unity 3D al *middleware* para indicar el nombre del usuario que inicia la comunicación. Con este dato el *middleware* genera un archivo de texto cuyo nombre está compuesto por el campo "UserName_" y dicho dato.

Figura 22. Estructura de un mensaje de control de clase nombre del usuario

- **Resultados de ejecución (Figura 23):** El campo "Clase" contiene la cadena de texto *FR*. Este mensaje se envía desde Unity 3D al *middleware* para indicar los resultados obtenidos tras su ejecución. El *middleware* extrae dichos resultados del campo "Datos" y los almacena en el archivo de texto generado al recibir un mensaje de control de clase **nombre de usuario**.

Figura 23. Estructura de un mensaje de control de clase resultados de ejercicio

- **Mensajes de error (Figura 24):** El campo "Clase" contiene la cadena de texto *ER*. Se envía desde el *middleware* a Unity 3D para indicar a este último, que ha habido un error que debe ser tratado antes de continuar con su ejecución. El tipo de error se indica en el campo de "Datos" existiendo los siguientes tipos:

  - ***AJ*:** El *middleware* no ha podido procesar el mensaje de articulaciones activas recibido e informa a Unity 3D de que debe realizar de nuevo el envío del mismo.

  - ***SP*:** El *middleware* no ha podido procesar el mensaje de solicitud de datos de movimiento recibido e informa a Unity 3D de que debe realizar de nuevo el envío del mismo.

  - ***NT*:** Cuando Kinect V2 no puede localizar ningún usuario de forma correcta informa al *middleware* mediante la modificación de una variable. Cuando el *middleware* lo detecta informa al usuario mediante un mensaje de error y, para que esta información sea también tenida en cuenta por Unity 3D, envía un mensaje de error indicándolo.

  - ***AT*:** Se envía siempre una vez se ha realizado el envío de un mensaje de error con el campo datos ***NT***, e, informa a la aplicación de que, tras haberse perdido la localización del usuario por parte de Kinect V2, esta, se ha vuelto a recuperar.

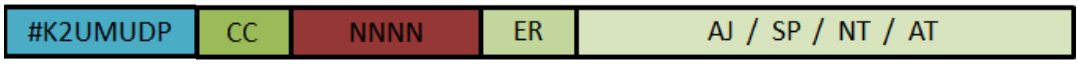


Figura 24. Estructura de un mensaje de control de clase mensajes de error

3.1.2.3. *Esquema de transmisión del proceso de comunicación*

Tras definir los distintos mensajes que utilizan el *middleware* y Unity 3D para gestionar el proceso de transmisión, se muestra este proceso esquemáticamente mediante un ejemplo del intercambio de mensajes el cual se lleva a cabo cuando se realiza una comunicación entre el *middleware* y Unity 3D.

En la figura 25 mostrada a continuación se muestra el esquema de transmisión esperado en una comunicación en la que los datos solicitados al *middleware* desde la aplicación son los asociados tanto a la posición como a la rotación captadas por Kinect V2.

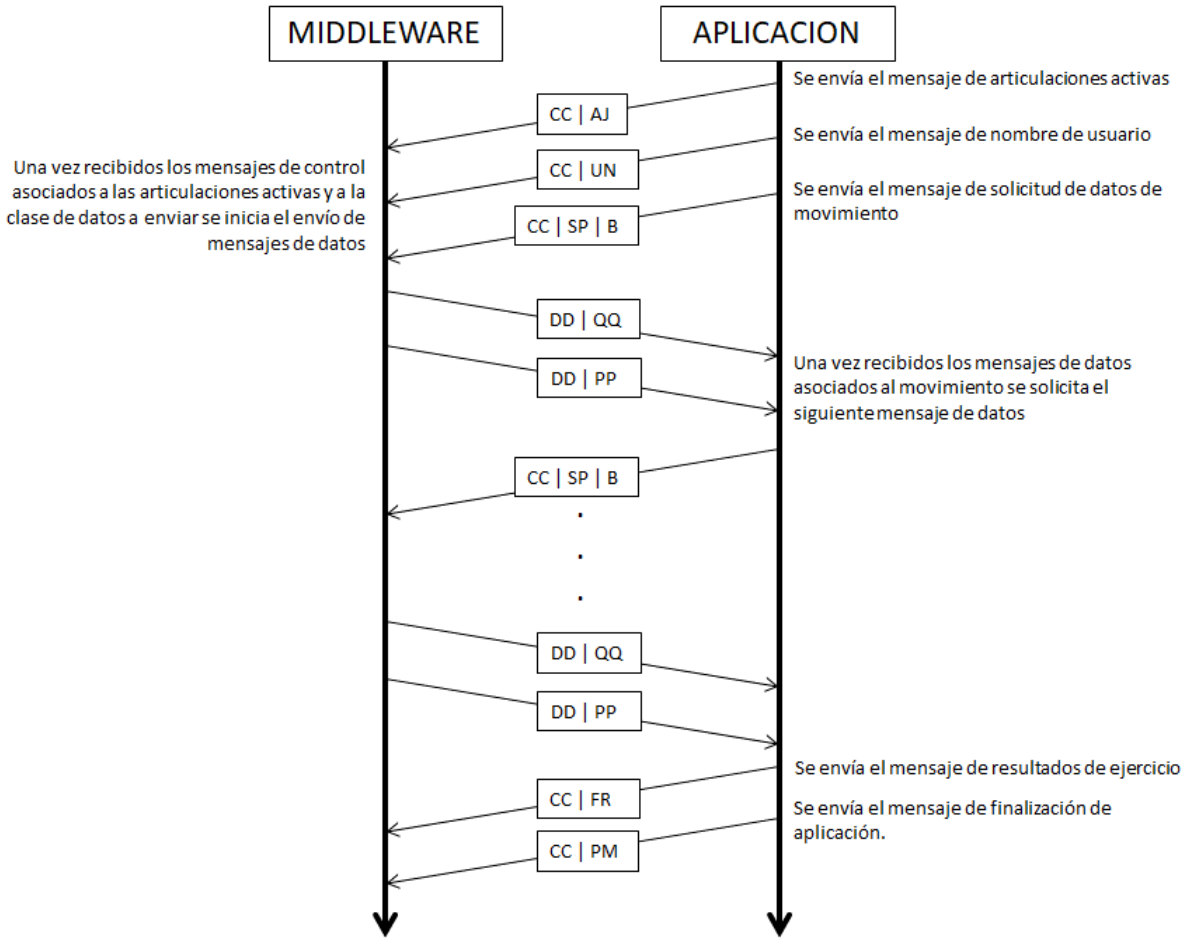


Figura 25. Esquema de transmisionesperado en una comunicación entre el middleware y la aplicación

Para comprobar que efectivamente se cumple este esquema de transmisión cuando el *middleware* y Unity 3D se comunican entre ellos, se simula una situación de comunicación como la descrita en la figura 25 y se analizan los paquetes UDP transmitidos en el proceso mediante la aplicación *WireShark*" [19].

Para ello se configura una comunicación en *localhost* en la que el *middleware* recibe por el puerto 8050 y emite por el puerto 8052, y la aplicación recibe por el puerto 8051 y emite por el puerto 8053.

Los resultados obtenidos se muestran en la figura 26.

No.	Time	Source	Destination	Protocol	Information	Text
1	0.000000	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=137	#K2UMUDPCC0121AJNeck|ElbowRight|ElbowLeft|ShoulderRight|
2	0.002006	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=20	#K2UMUDPCC0004UNUser
3	0.017041	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
4	0.943391	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=18	#K2UMUDPCC0002ERAT
5	0.944393	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=510	#K2UMUDPDD0494QQPBSpineBase:-0,011:0,019:1,734|SpineBase
6	0.966950	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
7	0.972461	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=510	#K2UMUDPDD0494QQPBSpineBase:-0,008:0,023:1,706|SpineBase
8	0.983988	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
9	0.999525	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
10	1.020576	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=509	#K2UMUDPDD0493QQPBSpineBase:-0,007:0,026:1,705|SpineBase
11	1.049145	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
12	1.074709	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=509	#K2UMUDPDD0493QQPBSpineBase:-0,003:0,028:1,711|SpineBase
13	1.099769	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
14	1.110794	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=508	#K2UMUDPDD0492QQPBSpineBase:0,000:0,030:1,716|SpineBase:
831	15.745474	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
832	15.780058	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=506	#K2UMUDPDD0490QQPBSpineBase:0,000:0,027:1,691|SpineBase:
833	15.811633	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
834	15.816646	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=506	#K2UMUDPDD0490QQPBSpineBase:-0,005:0,025:1,689|SpineBase
835	15.845215	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=17	#K2UMUDPCC0001SPQ
836	15.847219	127.0.0.1	127.0.0.1	UDP	8052 → 8051 Len=506	#K2UMUDPDD0490QQPBSpineBase:-0,009:0,023:1,688|SpineBase
837	15.863759	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=84	#K2UMUDPCC0068FREjercicio 1.- Troncos cortados 3 Tiempo
838	24.544444	127.0.0.1	127.0.0.1	UDP	8053 → 8050 Len=16	#K2UMUDPCC0000PM

Figura 26. Mensajes transmitidos en el inicio y finalizacion de una comunicación entre el middleware y la aplicación

Con estos resultados se comprueba que el esquema previsto en la figura 25 se cumple con exactitud, con la única discrepancia del mensaje número **4**, que indica que se ha recuperado la localización del usuario por parte de Kinect V2, de lo que el *middleware* informa a Unity 3D mediante un mensaje de error con el campo datos ***AT*** .

Además, también se puede comprobar lo siguiente:

- La información contenida en las cabeceras es la esperada para cada tipo y clase de mensaje enviado.

- La información numérica contenida en el campo de longitud de los datos coincide con la medida por el programa en el campo *Information* si le son restados los 16 *bytes* fijos que ocupan las cabeceras.

- Los puertos de entrada y salida de datos tanto del *middleware* como de Unity 3D coinciden con los configurados.

3.1.3. Interface gráfico del middleware

Para su utilización como aplicación independiente este *middleware* se implementa como una aplicación de escritorio directamente ejecutable por el usuario. A su vez está compuesta por varias ventanas cada una de ellas con una funcionalidad distinta. Desde ellas es posible desde iniciar, pausar o finalizar el *middleware*, hasta acceder a toda la información de control del funcionamiento del mismo. En los siguientes puntos se analiza el funcionamiento detallado de cada una de ellas.

3.1.3.1. Ventana de controles principales

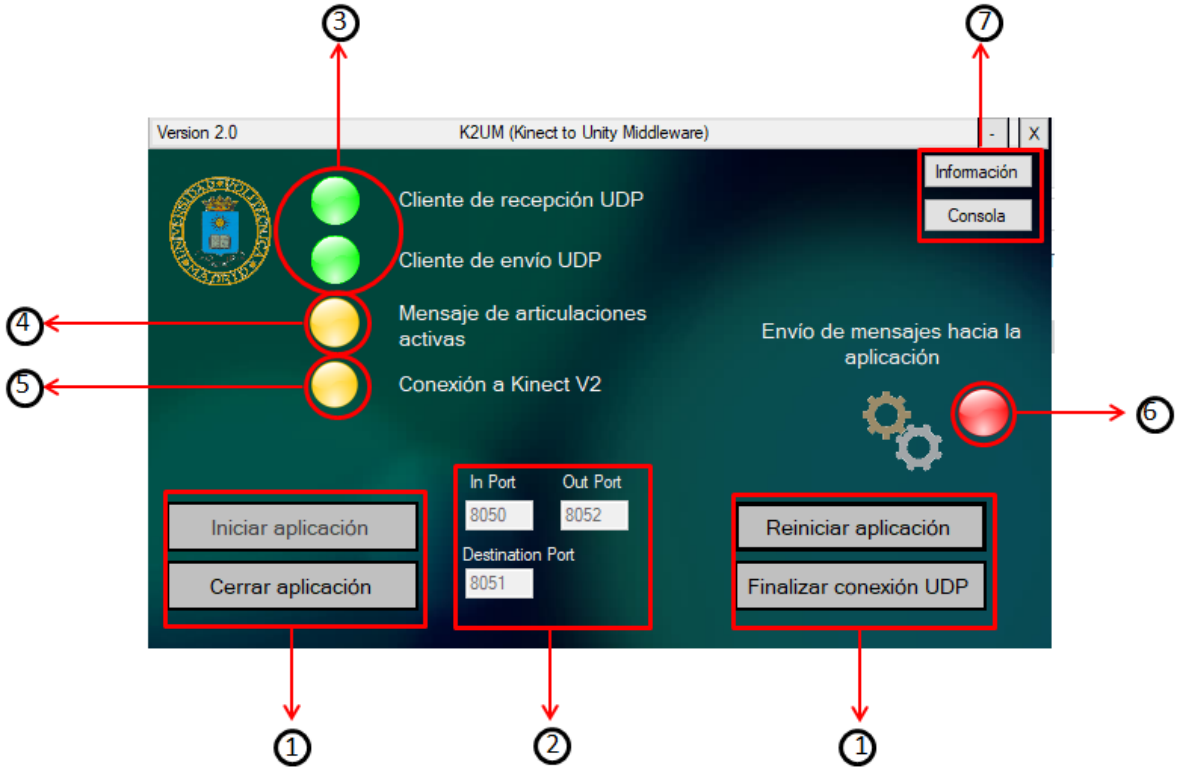


Figura 27. Ventana de controles del interface grafico del middleware

1. **Botones de control principal del *middleware*:** Rigen el funcionamiento principal de la aplicación, permitiendo su inicio, cierre y reinicio. También permiten finalizar únicamente la conexión UDP, liberando los sockets asociados a ella y permitiendo hacer cambios en los puertos configurados sin reiniciar el *middleware*.

2. **Puertos asociados a la comunicación UDP:** Los puertos de entrada, salida y destino que utiliza el *middleware* durante la creación de los sockets UDP. Pueden ser modificados únicamente cuando la comunicación UDP no ha sido todavía iniciada o ha sido finalizada previamente. Para facilitar su uso, tanto el *middleware* como Unity 3D poseen inicialmente una serie puertos predefinidos de manera que este valor no es necesario que sea introducido.

3. **Indicadores de conexión UDP:** Indican el estado de conexión a los clientes de envío y recepción UDP.

No se han iniciado los clientes UDP.

Los sockets están activos y listos para transmitir y recibir información.

4. **Indicador de mensaje de articulaciones activas:** Indica si se ha recibido un mensaje comunicando las articulaciones activas en este envío.

Ni se ha recibido ni se espera un mensaje con esta información.

El *middleware* está a la espera de recibir un mensaje con las articulaciones activas para iniciar el proceso de transmisión.

Ya se ha recibido este tipo de mensaje y se ha iniciado la transmisión.

5. **Indicador de conexión a Kinect V2:** Indica el estado de la conexión entre el *middleware* y el hardware Kinect V2.

No existe una conexión activa con Kinect V2.

Pese a existir una conexión activa con Kinect V2, esta no detecta ningún usuario de forma correcta.

Existe una conexión activa con Kinect V2 y esta ha detectado de forma correcta un usuario

6. **Indicador de estado de envío:** Indica el estado del envío de mensajes de tipo datos entre el *middleware* y Unity 3D.

No se está produciendo ningún envío.

El envío esta pausado tras haber finalizado su conexión la aplicación demandante de datos. Se mantiene hasta que se realice una nueva solicitud de datos.

Indica que el *middleware* y Unity 3D están intercambiando mensajes.

7. **Botones de cambio de pestaña:** Permiten la navegación entre las distintas pestañas del *middleware*. Son la pestaña de “Informacion” y la pestaña de “Consola”, y su contenido se muestra de forma detallada en los siguientes apartados.

3.1.3.2. Ventana de información

Esta ventana contiene información detallada relacionada con el envío y la recepción de datos vía UDP (Figura 28). Así, permite conocer desde el número de mensajes transmitidos en ambas direcciones hasta el número de errores producidos en este proceso. Desde ella, también es posible, acceder a información de control, como el contenido del campo "Datos" en los últimos mensajes enviados y recibidos, el número de articulaciones activas enviadas en esta transmisión, los datos relacionados con el último usuario conectado o los resultados obtenidos durante la ejecución de la última aplicación conectada.

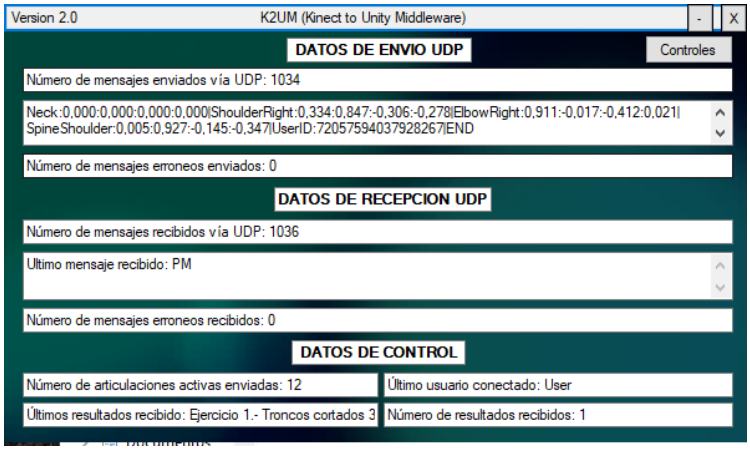


Figura 28. Ventana de información del interface gráfico del middleware

3.1.3.3. Ventana de consola

Esta ventana informa acerca de los procesos internos que realiza el *middleware* durante su ejecución (Figura 29). Cada vez que uno de los hilos lanzados realiza un proceso considerado de interés para el usuario, se añade un mensaje al texto mostrado en esta ventana, indicando que tipo de proceso se ha realizado y el momento en el que se ha llevado a cabo.

Toda la información mostrada en esta ventana se almacena en un archivo temporal llamado **`tempOut.info`** que, dado que actualmente no es necesario almacenar ningún dato del proceso de transmisión realizado, es eliminado cuando se finaliza la ejecución del *middleware*.

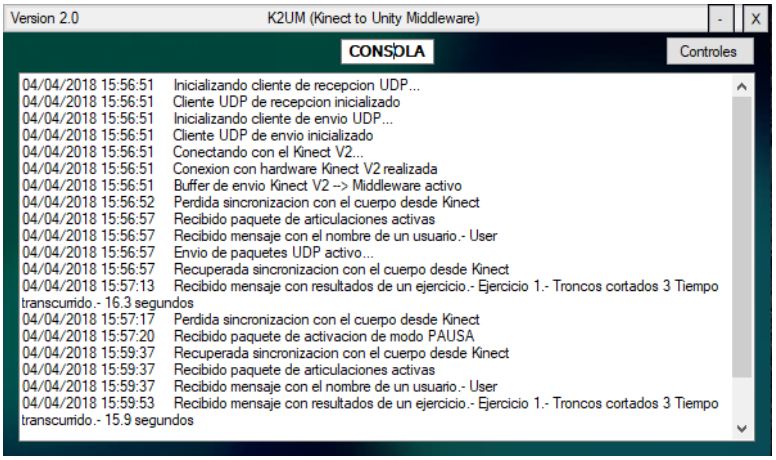


Figura 29. Ventana de consola del interface gráfico del middleware

3.2. El receptor de Unity 3D.

En este capítulo se describe como Unity 3D realiza el procesado de los datos recibidos vía mensaje UDP, de manera que, estos sean utilizados como sistema de control para un personaje implementado como un modelo 3D.

Dado que Unity 3D es únicamente un motor que controla la ejecución del videojuego. Todos los modelos 3D o imágenes que se utilicen deben ser importados desde otra aplicación de diseño gráfico como son Blender, 3ds Max, Maya o PhotoShop. En este caso, dado que es un software que permite su uso con licencia gratuita, es Blender el software elegido para el diseño del modelo 3D que se utiliza para el ejemplo práctico de este proyecto.

Como motor para videojuegos, Unity 3D permite al usuario modificar todas las características, tanto físicas como de comportamiento, que se pueden aplicar a los distintos objetos que componen una escena. Estas características van, desde el nivel de iluminación, la gravedad o sombreado aplicado al entorno, hasta definir de manera exhaustiva cualquier característica vinculada a los objetos presentes en la escena, como su velocidad, posición y rotación asociadas o, la forma en la que se relacionan dichos objetos unos con otros.

Para modificar estas características en tiempo de ejecución o definir estas relaciones entre los objetos que componen la escena, Unity 3D permite la ejecución de *scripts* codificados en distintos lenguajes de programación. Esta ejecución se realiza a través de *MonoDevelop*, una implementación de código abierto de *.NET Framework*, principalmente diseñada para C#. Así, este es el lenguaje utilizado para el desarrollo del *asset* de recepción de Kinect V2 y de las aplicaciones desarrolladas como muestra práctica.

Unity 3D llama *asset* [20] (Figura 30) a cualquier *item* representado que pueda ser utilizado en el videojuego o aplicación. Un *asset* puede provenir de un archivo que haya sido creado en una aplicación externa a Unity 3D, tal y como puede ser un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos multimedia que Unity 3D soporta. Unity 3D también permite crear sus propios *assets* y posteriormente exportarlos para su uso en futuras sesiones de trabajo.

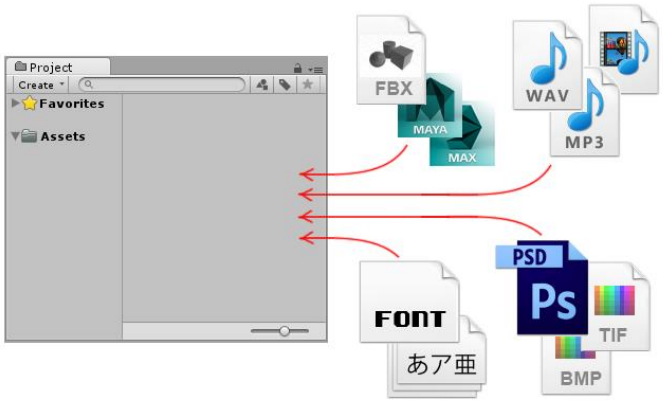


Figura 30. Tipos de Asset importables en Unity 3D

De esta manera se ha generado el *asset* que contiene los *scripts* que permiten la recepción de datos enviados desde Kinect V2 y el posterior procesado de estos para su utilización como control del videojuego. Para su identificación se ha exportado con el nombre de **`KinectAsset.unitypackage`** [21].

Para el acceso a este *asset*, una vez creada una sesión de trabajo en Unity 3D, este debe ser importado como *Custom Package* a la misma. Una vez hecho, en la carpeta `Assets` de la ventana `Project` aparece una nueva carpeta llamada `KinectAsset` con los *prefabs* y *scripts* asociados a este ítem (Figura 31).

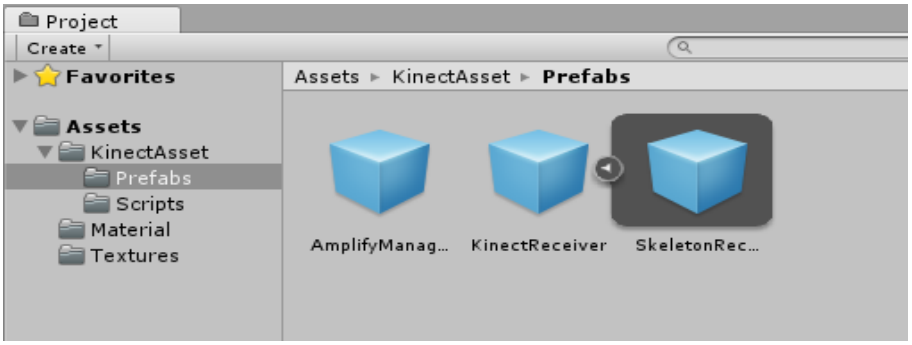


Figura 31. Objetos importados con el asset "KinectAsset.unitypackage"

Unity 3D llama *prefab* [22] a un objeto que se almacena con todas su características y *scripts* asociados, de manera que actúe como plantilla a partir de la cual se pueden crear nuevas instancias al objeto en la escena. Así, para el uso del *asset* que contiene todas las herramientas diseñadas, solamente es necesario arrastrar cada uno de los *prefabs* generados a la escena de trabajo actual.

3.2.1. KinectAsset.

Tal y como se muestra en la figura 31, este *asset* está compuesto por una carpeta con dos *prefabs* y otra con una serie de *scripts* que estos *prefabs* tienen asociados.

El primero de estos *prefabs* se llama *KinectReceiver* y contiene, por un lado, toda la lógica para la recepción y tratamiento de los datos enviados por el *middleware*, y por otro, el esqueleto receptor al que se aplican estos datos. El esqueleto receptor se muestra como otro *prefab* vinculado a *KinectReceiver* dado que su uso siempre se realiza de forma conjunta.

El otro *prefab* que compone este *asset*, llamado *AmplifyManager*, contiene toda la lógica necesaria para realizar la amplificación de los movimientos recibidos procedentes del *middleware*. Se facilita como *prefab* independiente de manera que, si no es desea amplificar el movimiento realizado por el usuario captado, no esté presente en la escena.

A continuación se pasa a analizar en profundidad estos elementos y su relación con el proceso de ejecución de la aplicación.

3.2.2. El esqueleto receptor: SkeletonReceiver

El *middleware* envía la información asociada al movimiento captado en forma de mensajes con el nombre de la articulación y la rotación o posición espacial que esta tiene. Para aplicar dicho movimiento en un modelo (avatar), *KinectAsset* utiliza un esqueleto receptor intermedio llamado *SkeletonReceiver*.

Este esqueleto receptor está compuesto por una serie de *GameObjects* [23] emparentados entre ellos con el mismo orden jerárquico que las articulaciones del esqueleto mostrado en la figura 9. El número de *GameObjects* es el mismo que el de articulaciones transmitidas por Kinect V2, de manera que estos actúan como articulaciones del esqueleto receptor. También se utiliza una herramienta de Unity 3D llamada *gizmos* para dibujar los huesos que unen dichas articulaciones. Esta herramienta permite la visualización de elementos gráficos de apoyo al diseño del videojuego, como son las líneas entre las articulaciones, que son mostrados en la ventana de diseño de la aplicación y no cuando esta se ejecuta. Esta herramienta es de gran utilidad ya que, aunque es necesario conocer la posición exacta del esqueleto receptor para su correcta colocación en la escena, este no debe ser visible durante la ejecución del videojuego.

En la figura 32 se muestra un objeto *SkeletonReceiver* posicionado en una escena de un proyecto en desarrollo de Unity 3D.

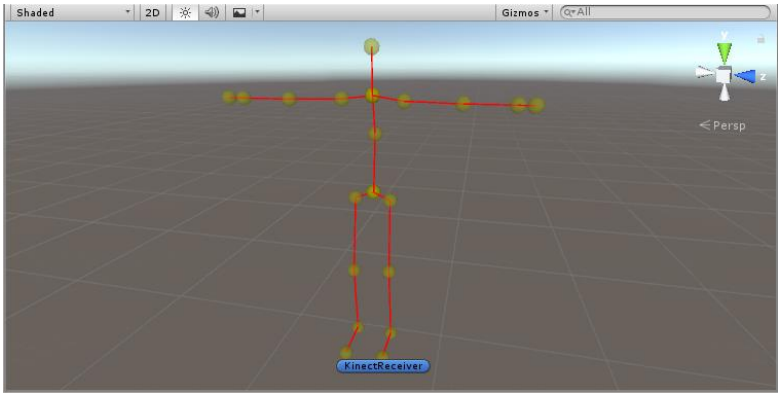


Figura 32. Esqueleto receptor asociado al objeto SkeletonReceiver

Si se realiza una comparación entre el esqueleto mostrado en la figura 32 y el mencionado anteriormente de la figura 9, aparecen ciertas discrepancias relacionadas con el número de articulaciones. Así, el esqueleto receptor tiene cinco articulaciones menos que el mostrado en la figura 9: 4 en cada brazo (en vez de 6) y dos en la cabeza (en vez de una). Las articulaciones no incluidas son las del final de la extremidad asociada, con lo que su rotación vinculada es nula.

Esto es importante debido a que el esqueleto receptor realiza toda la implementación del movimiento transmitido por el *middleware*, aplicando a cada articulación la rotación captada para ella en ese instante por Kinect V2. De esta manera, dado que el esqueleto receptor y el captado por Kinect V2 comparten la misma jerarquía de emparentamiento, se consigue que el movimiento realizado por ambos sea idéntico.

La figura 33 muestra un ejemplo práctico de este caso. Para comprobar la exactitud del proceso de implementación del movimiento, a la izquierda, se muestra también lo captado desde la aplicación de escritorio del SDK V2 de Kinect, *KinectStudio*.

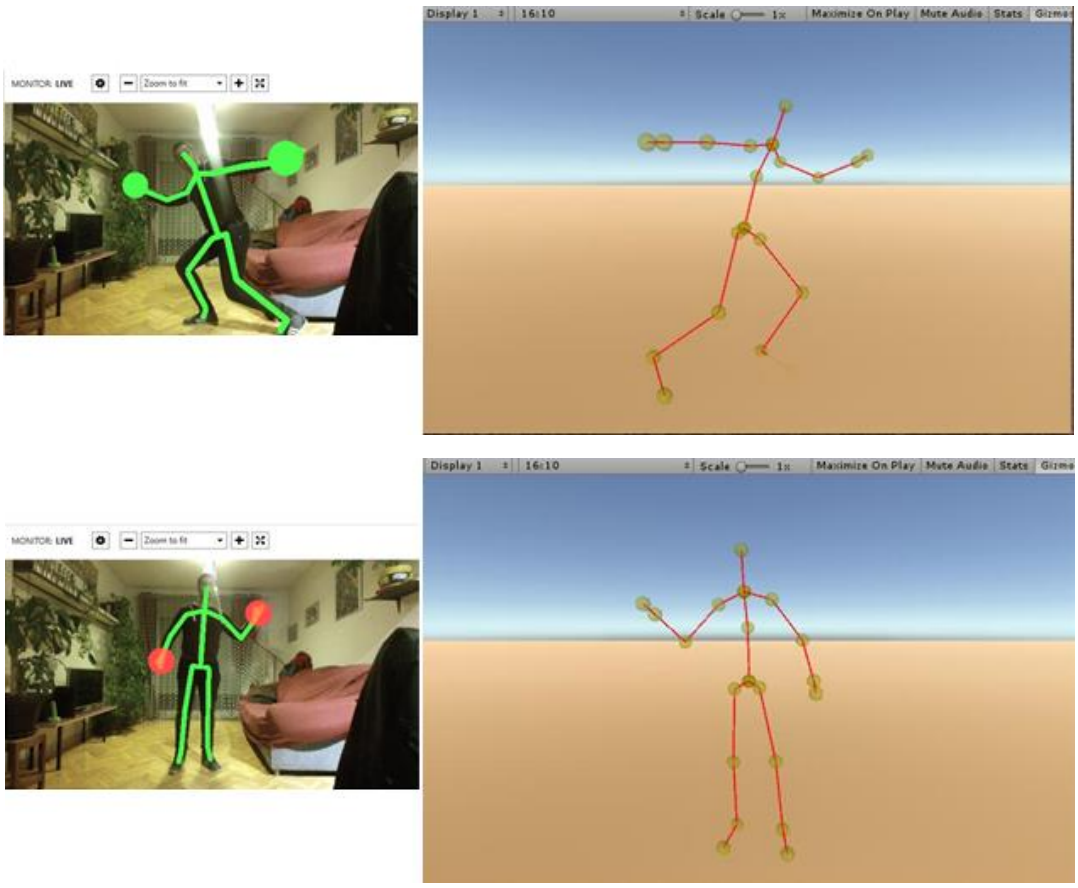


Figura 33. Captacion de movimiento reallzada por KinectStudio e implementación del movimiento transmitido por el middleware en el esqueleto receptor

Se puede comprobar como las posiciones de ambos esqueletos son las mismas, lo cual demuestra que los movimientos captados por Kinect V2 y transmitidos a través del *middleware* se reproducen con exactitud en el entorno de Unity 3D.

Cabe mencionar, antes de la finalización de este punto, que, de manera análoga al sistema de hilos que se mostraba para la ejecución del *middleware* en la figura 15, este *asset* se basa también en un sistema de funcionamiento en el que desde el hilo principal se generan tres hilos secundarios. De la misma manera que en el *middleware*, dos de ellos se encargan de la recepción y el envío de paquetes UDP, y un tercero se encarga de extraer la información asociada al movimiento de la cola de mensajes de datos recibidos y aplicarla al esqueleto receptor. Para la realización de estas tareas utiliza las clases las clases `UDPSend.cs`, `UDPReceive.cs` y `Rotation.cs`.

La única diferencia sustancial entre el sistema de hilos manejado en el *middleware* y el implementado en Unity 3D, es que, dado que Unity 3D no permite la creación de hilos de ejecución secundarios desde un hilo principal, es necesario implementarlos en forma de corrutinas [24].

Aunque a nivel de lo mostrado al usuario actúan de la misma manera, en realidad su comportamiento es muy distinto al de los hilos. A nivel de ejecución un hilo no es un concepto como tal, sino un hecho físico, de manera que, por cada núcleo del procesador, existen dos hilos que pueden acceder a parte de los recursos de uno de estos núcleos para llevar a cabo cálculos de manera independiente del hilo principal.

Sin embargo una corrutina es un concepto relacionado con la ejecución simultánea de varias aplicaciones compartiendo un mismo recurso. Para ello se ejecuta una tarea inicial (que en este caso es el hilo principal), y desde ella se inicia otra tarea definida como corrutina (en este caso los hilos secundarios), haciendo que de forma cíclica ambas tareas se cedan el control de procesador para continuar su ejecución desde el último punto en el que se encontraban.

Aunque la ejecución de las tareas no sea simultánea como en el caso de los hilos, el paso de una a otra es tan rápido, que el usuario no percibe este desfase. Así los resultados finales que se obtienen con ambas técnicas son muy similares (Figura 34).

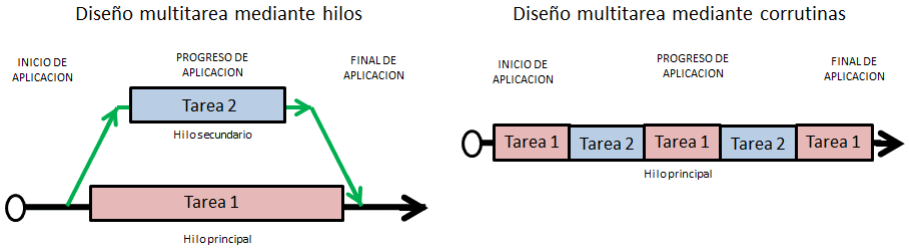


Figura 34. Descripción gráfica de un sistema multitarea basado en hilos y un sistema multitarea basado en corrutinas

3.2.3. El controlador: KinectReceiver

Una vez llevada a cabo la tarea de reproducir los datos de movimiento captados por Kinect V2, y transmitidos a través del *middleware*, el siguiente paso es conseguir que estos movimientos se apliquen en el esqueleto receptor de Unity3D y mediante este a un modelo 3D importado.

Para realizar esta tarea se tiene en cuenta que, todo modelo 3D que se genere con previsión de ser animado, es decir que se le aplique cierto movimiento de forma independiente para cada articulación, imprescindiblemente tienen que tener algún tipo de esqueleto asociado a él. De esta manera, a través del objeto *KinectReceiver*, es posible seleccionar cada una de las articulaciones que posee a dicho esqueleto y emparentarlas con las articulaciones del esqueleto receptor de Kinect V2.

Así, solo es necesario posicionar el esqueleto receptor de Kinect dentro del modelo 3D importado y desplazar sus articulaciones hasta la posición que ocupan en dicho modelo. Una vez iniciada la aplicación, el sistema de emparentamiento hace que, con cada movimiento del esqueleto receptor, se realice el mismo movimiento en la articulación del esqueleto del modelo

3D a la que está emparentada, y dado que, ambas ocupan la misma posición espacial, se realiza una transferencia óptima del movimiento del esqueleto receptor al modelo 3D importado [25].

Mediante esta técnica, es posible utilizar el esqueleto receptor implementado, en cualquier tipo de modelo 3D, independientemente de su esqueleto. Se puede desplazar cada una de las articulaciones que posee el esqueleto receptor, y es posible su usarlo en conjunto con cualquier modelo 3D, sin importar el tipo de fisionomía utilizada en su esqueleto interno p.ej. un personaje extraterrestre con articulaciones desproporcionadas como muestra la figura 35.

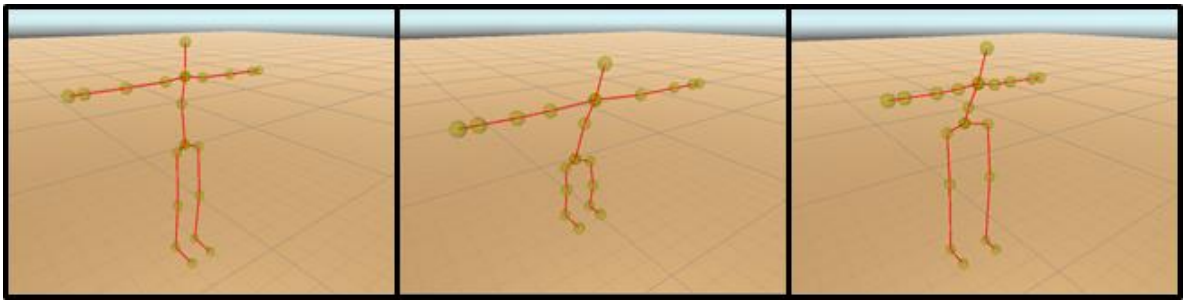


Figura 35. Muestra de las posible posiciones que pueden ser adopatadas por el esqueleto receptor

Para analizar el correcto funcionamiento de esta técnica de transferencia del movimiento de un esqueleto a otro, se realiza la descarga de un modelo 3D de prueba que imita un esqueleto humano. Esta descarga se realiza desde el *AssetStore* incluido en Unity 3D donde se pueden encontrar gran cantidad de modelos 3D gratuitos para su descarga [26]. El modelo 3D descargado se muestra en la figura 36.

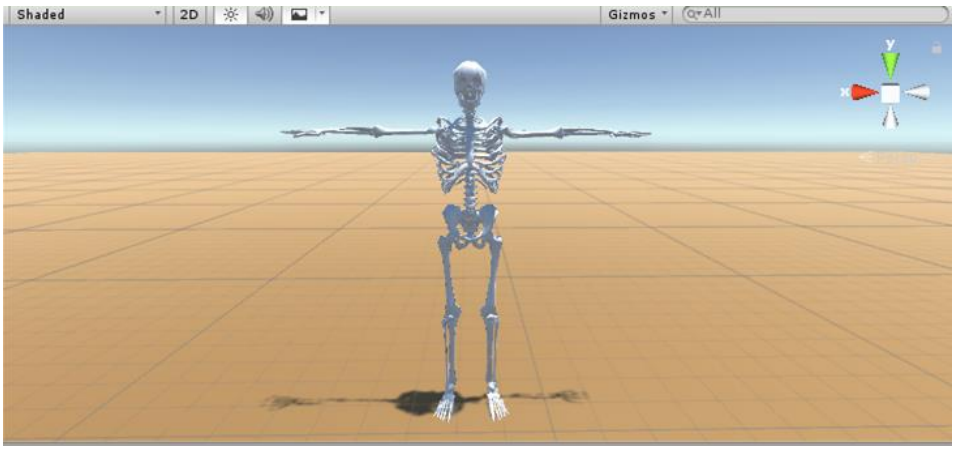


Figura 36. Modelo 3D de esqueleto descargado para su prueba en conjunto con KinectAsset

Este modelo se elige debido a que el número de articulaciones de su esqueleto asociado es suficiente como para solicitar al *middleware* todas las articulaciones existentes, de manera que, el análisis sea lo más exigente posible.

Para asociar ambos esqueletos en primer lugar se desplaza dentro de la escena el esqueleto receptor de manera que quede en el interior del modelo 3D, y posteriormente se desplazan las articulaciones hasta la posición que ocupan en él. En la figura 37 se muestra,

además del modelo 3D de la figura 36, el esqueleto receptor situado en su interior, coincidiendo las posiciones de cada articulación.

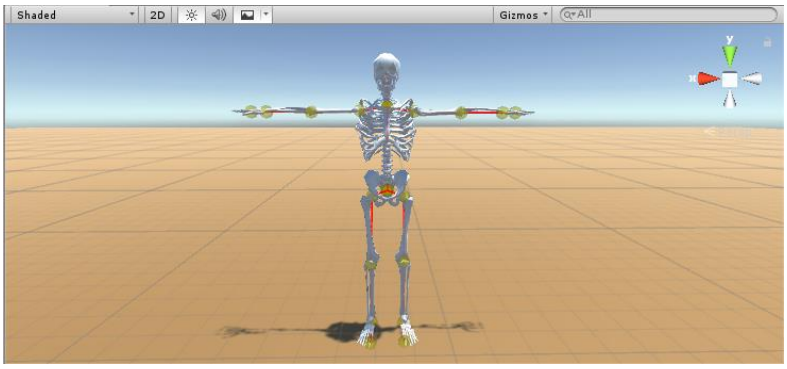


Figura 37. Modelo 3D de esqueleto descargado y esqueleto receptor de Kinect

Una vez posicionado el esqueleto receptor, solo resta indicar al *asset* el sistema de emparentamiento que se va realizar entre ambos esqueletos, asignando a cada articulación del esqueleto receptor su equivalente en el esqueleto del modelo 3D. El proceso para realizar este emparentamiento se explica con profundidad a continuación en este mismo apartado.

De manera análoga a lo mostrado en la figura 33, para comprobar la veracidad del movimiento transmitido, la figura 38 muestra, además del movimiento realizado por el modelo 3D en Unity 3D, lo captado desde la aplicación de escritorio del SKD v2 de Kinect, *KinectStudio*.

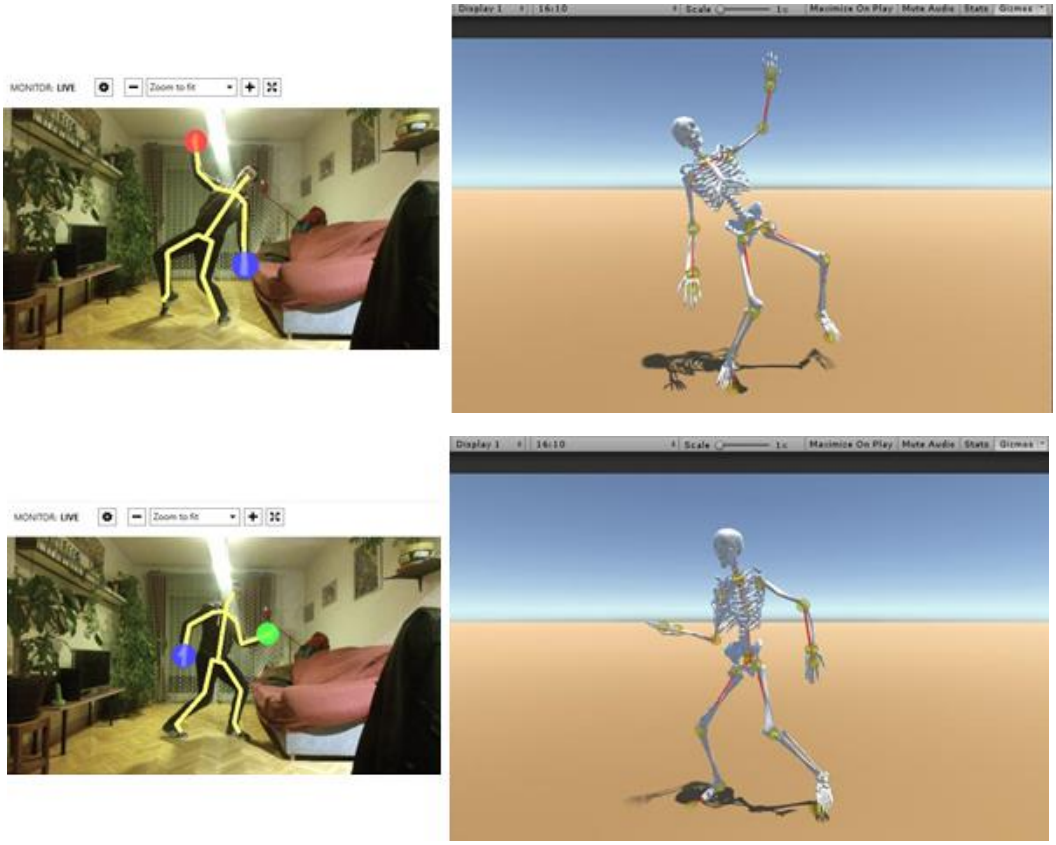


Figura 38 . Captación de movimiento realzada por KinectStudio e iImplementacion del moviento en el modelo 3D

Igual que en el caso mostrado en la figura 33, los movimientos realizados por el modelo 3D importado en Unity 3D, siguen fielmente a los realizados por el usuario y captados por Kinect V2. Esto demuestra que se ha conseguido realizar la tarea de transferir los movimiento del esqueleto receptor al modelo 3D.

Un punto antes mencionado antes sin profundizar, necesario para realizar la tarea de implementar el movimiento recibido en el modelo 3D, es el de como asociar cada una de sus articulaciones a sus equivalentes en el esqueleto receptor. Para ello Unity 3D permite la creación de variables públicas durante el desarrollo de un *script*, a las que se puede asociar objetos de la escena arrastrándolos a una casilla creada para ellas en el menú *Inspector* asociado al objeto.

En Unity 3D el menú *Inspector* [27] es utilizado para ver y editar las propiedades de un objeto y también preferencias y otros ajustes de configuración del software.

Cuando se selecciona un objeto o *GameObject*, el *Inspector* muestra las propiedades de todas las componentes asociadas a él, como son sus comportamientos físicos o las propiedades gráficas asociadas, y permite también editarlas. Como se ha comentado, cuando un objeto tiene un *script* asociado, las variables públicas de ese *script* son mostradas en el *Inspector* y pueden ser modificadas como cualquier otra propiedad de dicho objeto. Esto permite establecer, y modificar posteriormente, parámetros y valores de inicio de un *script* sin modificar el código.

En la figura 39 se muestra un ejemplo de un *Inspector* con varias componentes físicas asociadas y un *script* llamado "*Character*". En este script aparecen varias variables públicas de las que depende su ejecución, las cuales, pueden ser alteradas sin necesidad de modificar su código asociado.

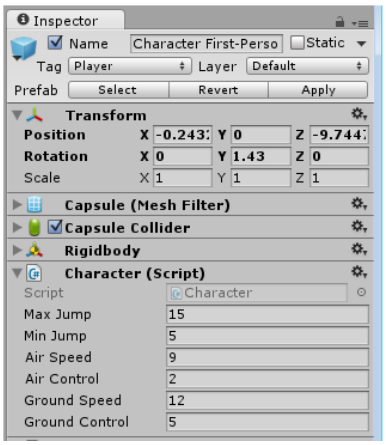


Figura 39. Ejemplo de Inspector en Unity 3D

Dado que la ejecución de este *asset* depende de una gran cantidad de *scripts*, la tarea de localizar todas las variables que deben de ser asignadas en cada uno de estos *scripts* sería muy compleja para alguien que no conociera su funcionamiento. Así, para simplificar este proceso, se ha utilizado una herramienta de Unity 3D que, mediante código, permite al usuario modificar el *Inspector* asociado al objeto y crear uno propio que se ajuste a sus necesidades de uso.

La figura 40 muestra el *Inspector* inicial que se muestra al seleccionar el objeto *KinectReceiver* en la escena.

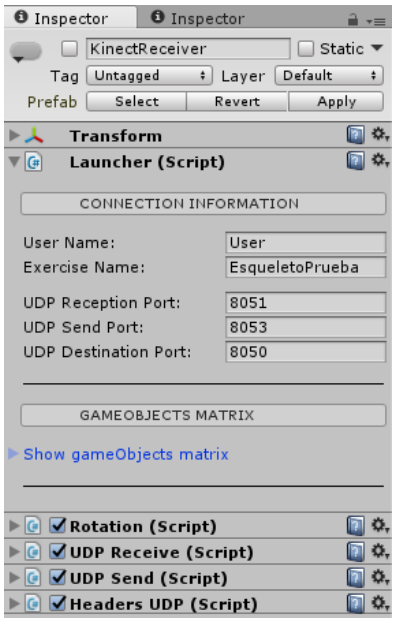


Figura 40. Ventana de inspector asociada a un objecto KinectReceiver

Tal y como se comentaba anteriormente, el *Inspector* de este objeto muestra todos los scripts que tiene asociados. El primero de todos ellos, llamado *Launcher*, actúa como controlador al contener todas las variables públicas a las que el resto de *scripts* acceden y que deben ser asignadas por el usuario de forma previa a la ejecución. También tiene asociado un *script* que modifica su aspecto y comportamiento para simplificar el proceso de configuración y utilización del *asset*.

Visualmente, el *Inspector* asociado al objeto *KinectReceiver*, está dividido en dos zonas. La primera de ellas, situada en la parte superior, permite al usuario introducir la información asociada a la comunicación con el *middleware*. En ella se definen, tanto los campos con el nombre del usuario y del ejercicio, como los puertos asociados a los *sockets* por los que se realiza la comunicación UDP. El nombre del usuario y del ejercicio se envían al *middleware* en forma mensajes de control de clase **nombre de usuario (UN)** y **resultados de ejecución (FR)** respectivamente,

La siguiente zona del *Inspector*, situada bajo la anterior, es donde el usuario puede asociar cada una de las articulaciones del modelo 3D importado a Unity 3D con las de su equivalente del esqueleto receptor. La figura 41 muestra una ampliación de la ventana asociada a esta zona.

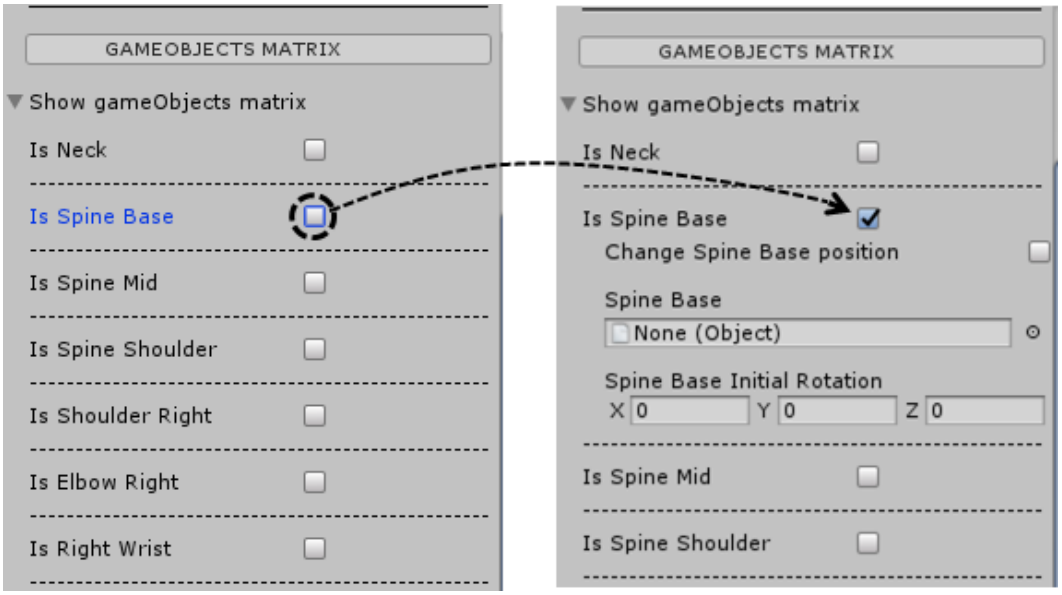


Figura 41. Matriz de objetos de una ventana de inspector asociada a un objecto KinectReceiver

En esta ventana se muestra una lista con todos los nombre de las articulaciones asociadas al esqueleto de recepción, todos ellos vinculados a una casilla de activación situada en su lateral. Al pulsar sobre esta casilla de activación se abre el desplegable mostrado en la parte derecha de la figura 41. Una vez abierto el desplegable, todas las articulaciones comparten la misma estructura de presentación de variables, con un campo vacío bajo su nombre para arrastrar el objeto asociado a la articulación equivalente en el modelo 3D, y otro para modificar su rotación inicial asociada en los tres ejes axiales. Este último campo es de utilidad para casos en los que el modelo 3D tenga menos articulaciones en sus extremidades que el esqueleto receptor. Un ejemplo de esto es un modelo que solo tenga una articulación en el brazo, en este caso el hombro, y además que porte algo al final de esta extremidad. Con este campo es posible forzar la rotación intrínseca que aplicaría el codo si estuviera presente en el modelo, para que, el objeto portado al final del brazo no se muestre rotado sobre su eje.

En la figura 41 se muestra la articulación asociada a *SpineBase* como ejemplo, debido a que esta es la única que tiene una característica adicional en el desplegable que se muestra al pulsar sobre su casilla de activación en el *Inspector*. Esta característica permite indicar al *asset* si se desea que este cambie la posición espacial de la articulación *SpineBase* de la misma manera que lo hace la del usuario captado por Kinect V2. Con esta opción desactivada se consigue dejar fijo el modelo 3D en un punto del espacio aunque el usuario captado se desplace. Esto es útil en aplicaciones donde el modelo permanece siempre en la misma posición y solo interesa captar el movimiento asociado a una o varias articulaciones.

3.2.4. El amplificador de movimiento: AmplifyManager

El último *prefab* contenido dentro del asset *KinectAsset* es el *AmplifyManager*. Este objeto tiene asociado una serie de *scripts* que permiten amplificar la rotación llevada a cabo por cualquiera de las articulaciones del esqueleto receptor y aplicar esta rotación amplificada a la articulación equivalente del modelo 3D.

Para llevar a cabo esta amplificación, la primera idea que surge es, tomar directamente la rotación asociada a la articulación a amplificar y operar con ella parar lograr el movimiento deseado. Sin embargo esta idea es descartada, debido, por un lado, a los problemas que aparecen al ser tenidas en cuenta todas las dependencias existentes entre las articulaciones, y por otro, a los complejos cálculos que la aplicación necesitaría realizar para llevar a cabo para la implementación del movimiento.

Así, dado que la amplificación directa del ángulo de rotación de una articulación se torna demasiado compleja, se opta por el uso de una técnica distinta para la implementación de una rotación sobre la articulación de un modelo 3D, llamada cinemática inversa [28].

Hasta ahora todos los movimientos realizados por el esqueleto receptor se producen mediante la rotación de los ángulos de las articulaciones a determinados valores. La posición de una articulación "hija" cambia de acuerdo a la rotación de su "padre", y de esta forma se puede determinar el punto final de una cadena de articulaciones a partir de los ángulos y las posiciones relativas de las articulaciones individuales que contenga. Este método de hacer rotar las articulaciones de un esqueleto es conocido como cinemática directa.

Sin embargo, existe la posibilidad de afrontar la tarea de hacer rotar las articulaciones de un esqueleto desde un punto de vista opuesto. Consiste en localizar una posición en el espacio y trabajar hacia atrás para encontrar una forma válida de orientar las articulaciones haciendo que el punto final de la extremidad termine en esa posición. Este enfoque es conocido como cinemática inversa o IK (Figura 42).

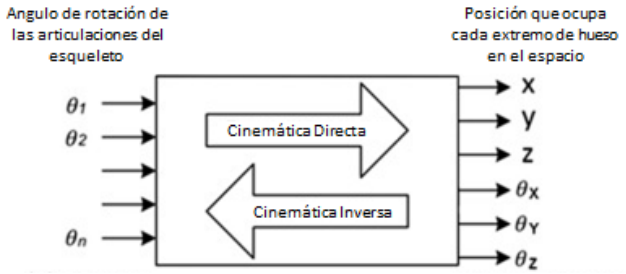


Figura 42. Descripción gráfica de los enfoques de cinemática directa e inversa

Así, mediante un *script* asociado al objeto *AmplifyManager* llamado `IKManager.cs` [29], es posible aplicar esta técnica a una articulación concreta seleccionada, consiguiendo que su rotación dependa de la posición en el espacio de un *GameObject* objetivo al que siempre apunta

Una vez conseguido esto, el siguiente paso es conocer el ángulo de rotación de la articulación a amplificar. Dado que se desea implementar una rotación en tres dimensiones, es necesario conocer tanto el ángulo de rotación del eje horizontal como el del eje vertical de la articulación en cuestión. Para ello este *prefab* tiene emparentada una carpeta con cuatro *GameObject* en su interior llamados calibradores. El primero de ellos, llamada *JointEdge*, debe situarse en el eje de giro de la articulación, es decir ella misma, el siguiente, cuyo nombre es *EndOfBone*, en el elemento que se desplaza, habitualmente el extremo del hueso unido a la

articulación, y dos últimos, llamados *VerticalOrigin* y *HorizontalOrigin*, que deben se situarse en los puntos donde se considere que los ángulos de rotación vertical y horizontal de la articulación son cero en cada caso.

La figura 43 muestra de forma gráfica este proceso.

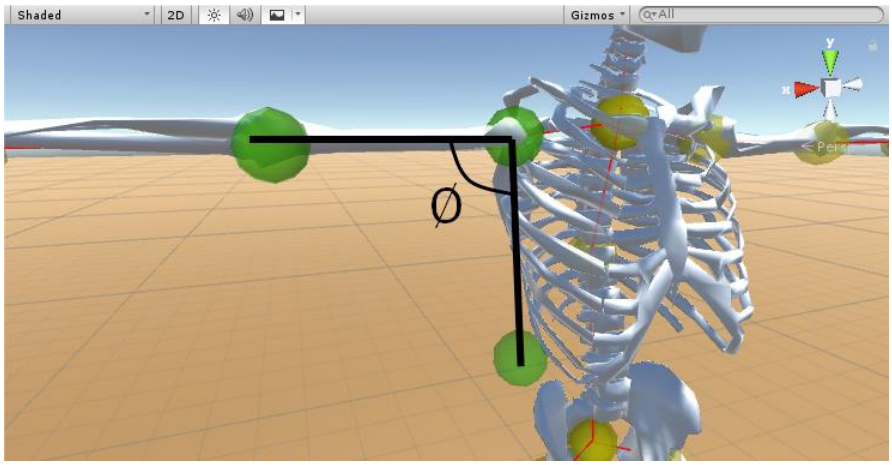


Figura 43. Medición del grado de rotación vertical de un hombro mediante tres calibradores

Con los ángulos de rotación vertical y horizontal ya calculados el siguiente paso es relacionar estos ángulos con la posición espacial del *GameObject* objetivo generado con el *script* de control de cinemática inversa. Para ello solo es necesario conocer los valores máximos y mínimos que tienen tanto los ángulos de rotación como las posiciones espaciales para cada eje del objetivo y generar un sistema de ecuaciones que los relacione.

La figura 44 ilustra de forma gráfica este proceso.

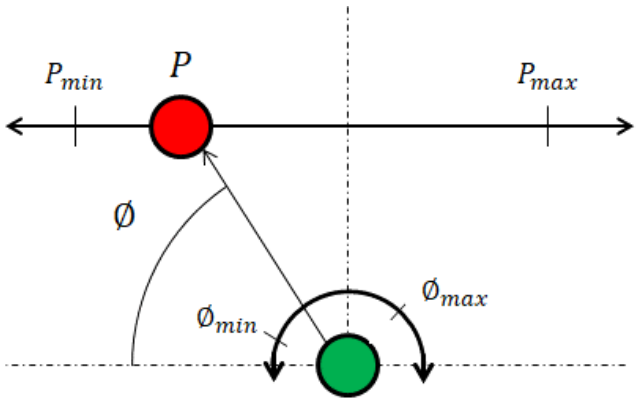


Figura 44. Proceso de conversion de una rotacion en una traslacion en un eje

Dado que es un sistema de ecuaciones lineales se puede asumir que:

$$Ø = \frac{x\,P}{y}$$

Siendo $Ø$ el ángulo medido y P la posición espacial del objeto, ambos para ese eje en concreto, y x e y los valores de ajuste a aplicar para la conversión.

De esta manera si conocemos los valores máximos y mínimos del ángulo medido y de la posición espacial del objetivo de la cinemática inversa es posible calcular los valores de x e y.

$$\emptyset_{min} = \frac{x\,P_{min}}{y} \qquad \emptyset_{max} = \frac{x\,P_{max}}{y}$$

$$x = \frac{(P_{max}\emptyset_{min}) - (P_{min}\emptyset_{max})}{P_{max} - P_{min}}$$

$$y = \frac{\emptyset_{max} - x}{P_{max}}$$

El método para el cálculo de estos valores máximos y mínimos se indica posteriormente en este apartado.

Estas fórmulas además de permitir la transformación de un ángulo medido a una posición en un eje axial, también son la clave para la amplificación del movimiento. Esto es así debido a que, una vez conocido el ángulo máximo de la articulación y su punto en el espacio asociado para el objetivo de la IK, solo es necesario disminuir el valor de este ángulo máximo, manteniendo igual las variables *x* e *y*, para que, cuando el ángulo medido por los calibradores sea el nuevo ángulo máximo, el objetivo de la cinemática inversa se desplace hasta su máxima posición en el eje, amplificando con ello la rotación asociada a la articulación (Figura 45). Esta tarea se realiza a través de la clase **`Amplifier.cs`.**

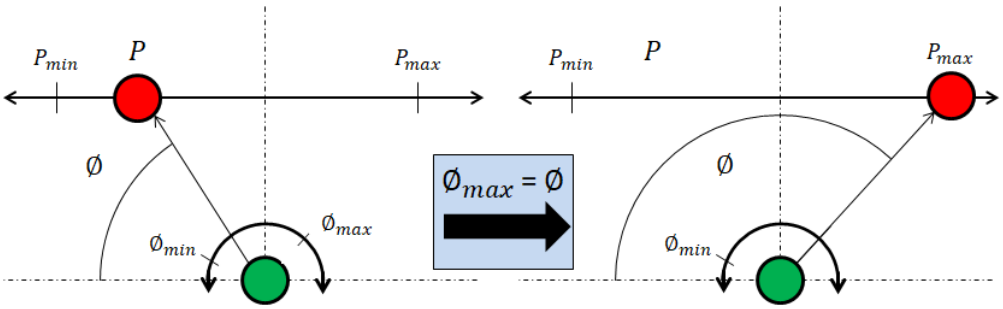


Figura 45. Mecanica del sistema de amplificación basado en cinemática inversa

Una vez diseñado el sistema de amplificación al completo solo es necesario calcular el grado de amplificación necesario a aplicar en cada caso. Este cálculo depende en última instancia del programador que vaya a usar la herramienta ya que puede variar según las necesidades de la aplicación a implementar. En este caso se ha optado por dotar al amplificador de un temporizador que, durante unos segundos, calcula el valor máximo del ángulo medidos por los calibradores asociados a la articulación a amplificar, y sustituye por él el ángulo máximo del sistema, tal y como muestra la figura 45.

Una vez más, como muestra práctica del proceso descrito en este apartado, se ha probado su funcionamiento sobre el hombro derecho del modelo 3D utilizado para un propósito similar en el apartado anterior de este documento, aplicando únicamente al eje vertical la amplificación. Para esta tarea, el ángulo máximo que el usuario de prueba alcanza durante la calibración es aproximadamente la mitad del ángulo que está configurado como máximo posible en Unity 3D.

Los resultados se muestran a continuación en la figura 46.

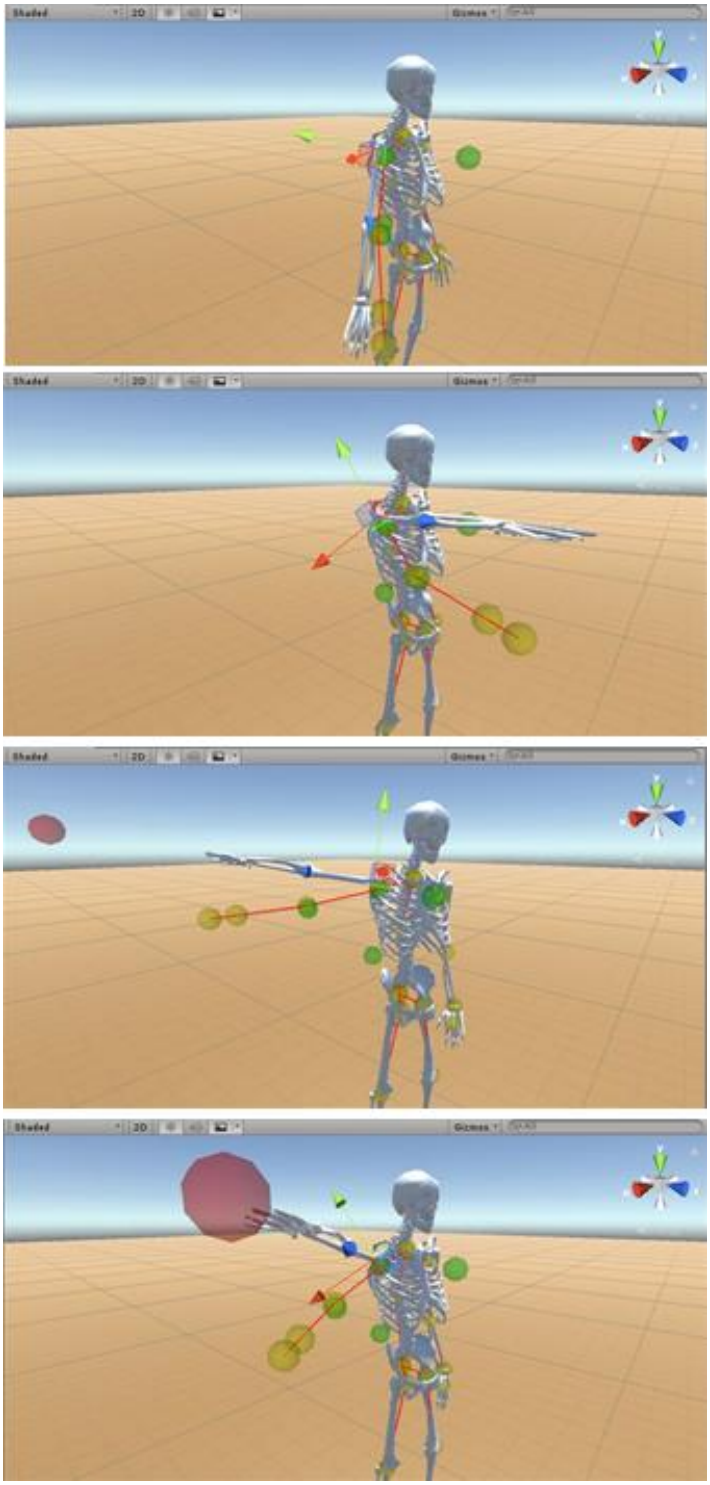


Figura 46. Efecto de la amplificacion sobre la articulacion de un modelo 3D

En esta figura se puede comprobar cómo, mientras que todas las articulaciones no amplificadas del modelo 3D siguen fielmente las rotaciones de sus equivalentes del esqueleto receptor, su hombro derecho mantiene únicamente la misma rotación horizontal, siendo, sin embargo, muy superior la rotación vertical que alcanza.

Por último, de la misma manera que el objeto *KinectReceiver* tiene un *Inspector* propio para facilitar su configuración, el objeto *AmplifyManager* también posee el suyo.

La figura 47 muestra el *Inspector* que aparece al seleccionar este objeto en la escena.

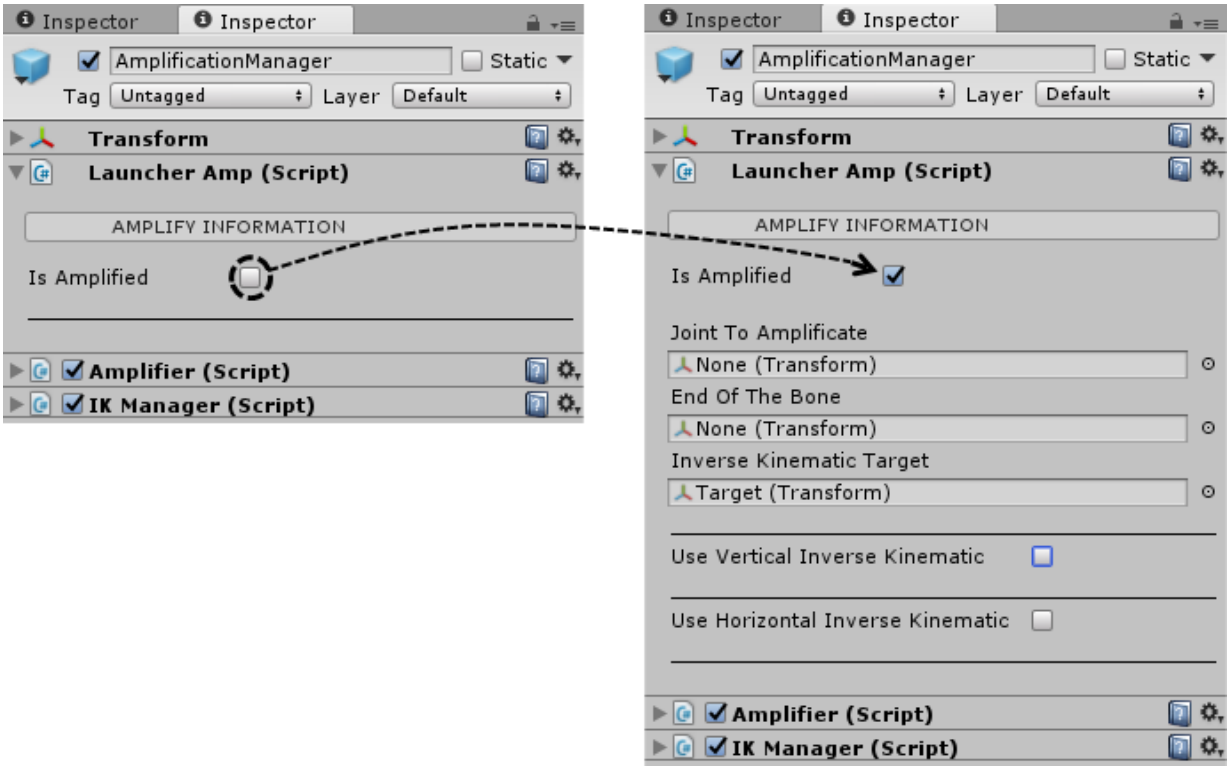


Figura 47. Ventana de inspector asociada a un objecto AmplifyManager

Una vez seleccionada dentro del *Inspector* la opción que permite el uso de la amplificación, el siguiente desplegable muestra la información inicial necesaria para realizar este proceso. El primer campo llamado "*Joint To Amplificate*" es donde el usuario tiene que arrastrar el *GameObject* asociado a la articulación del modelo 3D a amplificar. El siguiente tiene como nombre "*End Of The Bone*", y es donde el usuario debe arrastrar el *GameObject* asociado al final del hueso que se desplazada cuando rota la articulación amplificada.

El campo "*Inverse Kinematic Target*" se muestra con un *GameObject* ya asociado desde su inicio e indica al *asset* cuál es el objetivo al que el sistema de cinemática inversa debe hacer mirar el *GameObject* asociado a la articulación a amplificar. De esta manera, al desplazar este objetivo por la escena se modifica la rotación del *GameObject* asociado al campo "*Joint To Amplificate*", y con ello la posición espacial del *GameObject* asociado al campo "*End Of The Bone*".

Los siguientes desplegables permiten la activación de la cinemática inversa para cada uno de los ejes vertical y horizontal. Dado que su contenido es idéntico, solamente se analiza uno de los dos apartados. El contenido se muestra a continuación en la figura 48.

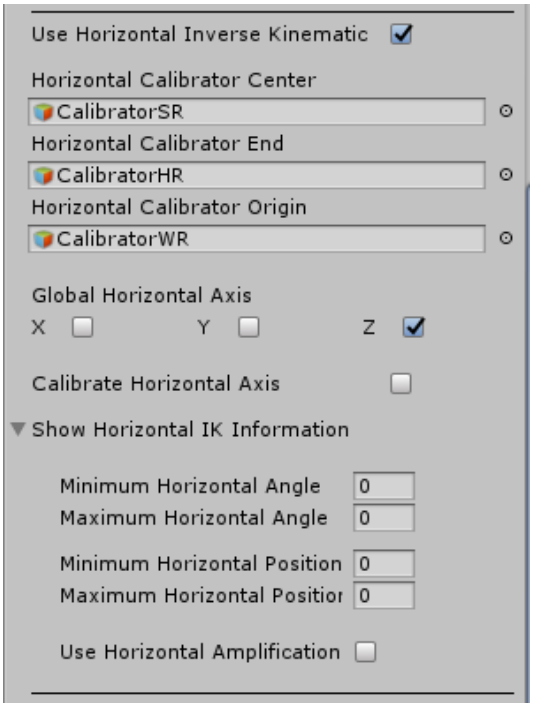


Figura 48. Configuracion de la cinematica inversa del eje horizontal de una ventana de inspector asociada a un objecto AmplifyManager

Una vez activada la opción de uso de la cinemática inversa en uno de los ejes de control, se muestra un desplegable con sus opciones de configuración asociadas.

En primer lugar aparecen los campos para asociar los *GameObjects* con calibradores de este eje, vinculados ya de forma automática a los *GameObjects* con el mismo nombre incluidos en el *prefab* y que pueden ser modificados a decisión del usuario. A continuación aparece una casilla para que el usuario indique cuál de los ejes de abscisas desea que actúe como eje horizontal.

La siguiente opción que aparece en el *Inspector* es "*Calibrate Horizontal Axis*". Está asociada a la calibración de este eje y es de vital importancia ya que permite al diseñador de la escena el cálculo de los valores máximos y mínimos para el ángulo de la articulación y la posición espacial del objetivo de la IK, es decir, los valores asociados a las variables $\emptyset_{min}, \emptyset_{max}, P_{min}$ y P_{max} definidas en la figura 44.

Una vez activada esta opción, Unity 3D calcula en tiempo de ejecución estos valores y los muestra al usuario en dos casillas tal y como se muestra en la figura 49. En el ANEXO II de este documento se documenta de forma detallada el mecanismo para la realización de este proceso.

Figura 49. Calibracion de los valores maximos y minimos de la cinematica inversa del eje vertical

Con esta casilla activada, el diseñador puede ejecutar la aplicación dentro de Unity 3D y desplazar por la escena el *GameObject* asociado al campo "*Inverse Kinematic Target*" hasta que las posiciones, tanto iniciales como finales, del extremo del hueso asociado a la articulación del modelo 3D sean la deseadas para todo su movimiento. Cuando se hayan localizado los puntos iniciales y finales, se copian los valores de posición y ángulo mostrados por el *Inspector* en los campos "*Minimum Horizontal Angle*", "*Maximum Horizontal Angle*", "*Minimum Horizontal Position*" y "*Maximum Horizontal Position*" mostrados en la figura 48.

La casilla "*Use Horizontal Amplification*" permite que, además de utilizarse el sistema de cinemática inversa, se active en este eje la amplificación.

3.3. La implementación práctica: El proyecto *BLEXER*

Una vez realizado el análisis técnico del sistema del comunicación que habilita el control de un modelo 3D en Unity 3D mediante Kinect V2, y la amplificación de los movimientos que dicho hardware capta, el último paso es generar una aplicación general desde Unity 3D, donde se implementen todas las características descritas hasta ahora.

Tal y como se ha comentado en los primeros apartados de este documento, el motivo de la realización de este proyecto de fin de grado, es su implementación en el proyecto *BLEXER* como herramienta para posibilitar el desarrollo de videojuegos para personas con diversidad funcional en el entorno de diseño Unity 3D. De esta manera, el ejecutable generado se ha basado en los cuatro ejercicios clásicos que este proyecto tenía ya diseñados, implementado en ellos los mismos movimientos de rehabilitación y añadiendo algunas características que aumentan su jugabilidad.

En la figura 50 se muestra la ventana de inicio de este ejecutable.

Figura 50. Ventana de inicio del ejecutable "Phiby´s Adventure Unity 3D Demo Version"

En esta ventana se muestran, por un lado, unos controles básicos que permiten al usuario introducir su nombre y decidir si desea que se active el amplificador de movimientos, y por otro, cuatro botones que llevan a cada uno de los ejercicios implementados.

Las limitaciones temporales no han permitido crear modelos 3D propios para esta versión de demostración, de manera que, para el desarrollo de los ejercicios han sido utilizados modelos 3D gratuitos descargados desde el *AssetStore* e importados desde la versión anterior de la aplicación. La configuración del asset de Unity 3D que permite el control del juego desde Kinect V2 se ha realizado siguiendo los pasos mostrados en el ANEXO II.

El diseño de todos los ejercicios se ha llevado a cabo teniendo en cuenta que están dirigidos a usuarios que pueden poseer cierto grado debilidad muscular, haciendo así necesaria la amplificación de los movimientos que estos realizan. Para ello, se tienen en cuenta las necesidades que muestra cada jugador, midiendo el grado de movilidad máximo de la articulación implicada en el movimiento durante los primeros segundos de juego.

El funcionamiento y las características operacionales de cada uno de estos cuatro ejercicios son detallados en los apartados que se muestran a continuación. Además, también se muestra el ejercicio de la versión anterior en el que está basado y sus diferencias de funcionamiento.

3.3.1. Ejercicio 1.- ¡Corta el tronco!

Este ejercicio está basado en el ejercicio de la primera versión del videojuego *Phiby´s Adventure* "*Chop the wood*" (Figura 51). La mecánica del ejercicio es básicamente la misma en ambas versiones con la única diferencia de que en la primera versión no aparecían distintos tipos de troncos.

Figura 51. Ejercicio de referencia de la primera version de Phiby´s Adventure para el ejercicio 1 - ¡Corta el tronco!

- **Objetivo:**

El usuario debe cortar un número de troncos indicado levantando para ello el brazo hasta cierta altura, en la que una esfera se ilumina y, tras mantener esa posición durante los instantes necesarios para que la esfera se ilumine por completo, bajar el brazo hasta impactar con el tronco (Figura 52).

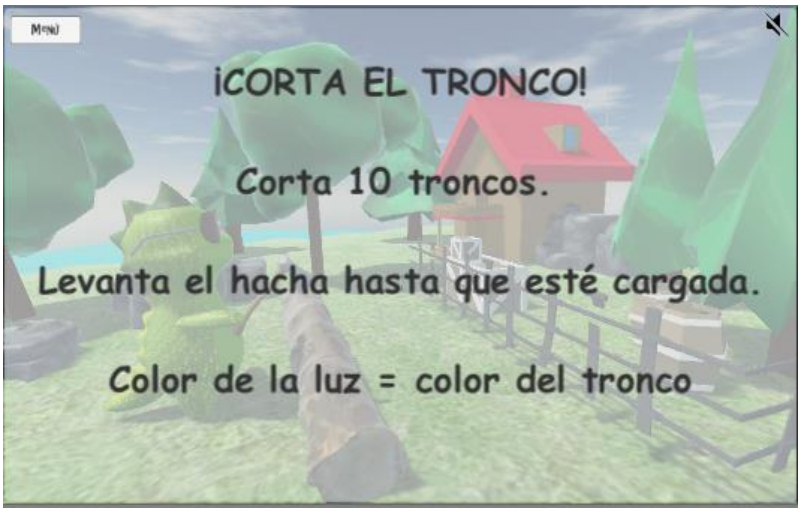


Figura 52. Ventana de inicio del ejercicio 1 - ¡Corta el tronco!

Este ejercicio se ha diseñado para ejercitar los músculos del hombro. Está pensado como un ejercicio con pocas repeticiones del movimiento ya que la posición superior ha de ser mantenida un cierto tiempo (Figura 53).

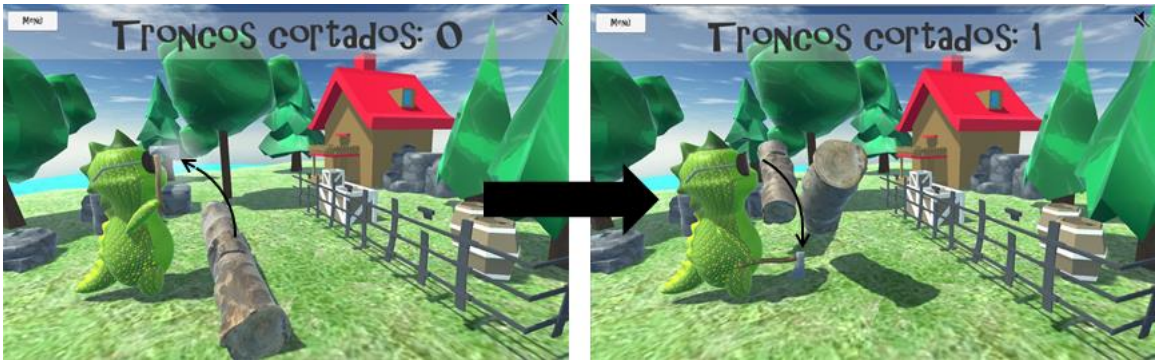


Figura 53. Mecánica de juego del ejercicio 1 - ¡Corta el tronco!

- **Dificultad:**

La esfera de iluminación es la encargada de indicar al usuario cuando ha transcurrido el intervalo de tiempo necesario con el brazo en alto, que permite el corte del tronco. Sin embargo, esta esfera tiene tres ciclos de iluminación, pasando del color blanco al verde, y posteriormente al rojo. Estos colores son debidos a que algunos troncos también son de otro color y es necesario esperar hasta que la esfera llegue a él para poder cortarlos (Figura 54).

El terapeuta puede controlar la intensidad del ejercicio a través de la modificación del número de troncos que el usuario debe de cortar, el tiempo que debe de mantener el brazo en alto para que se habilite el corte del tronco y la probabilidad de que aparezcan troncos de otros colores.

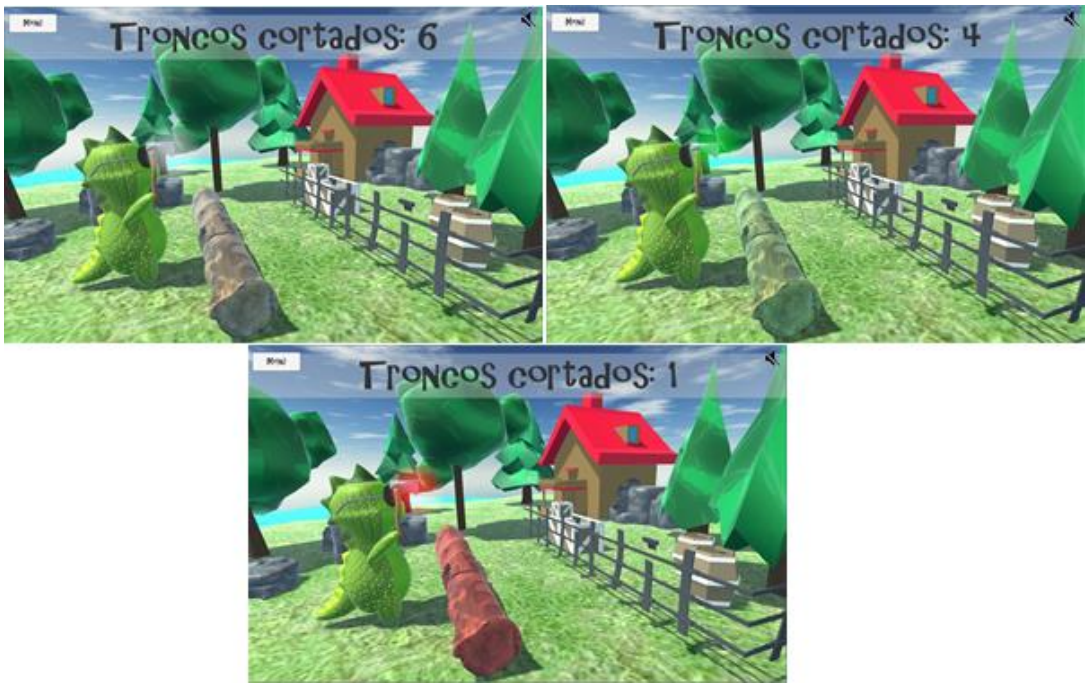


Figura 54. Distintos troncos objetivo del ejercicio 1 - ¡Corta el tronco!

3.3.2. Ejercicio 2.- ¡Trepa el arbol!

Este ejercicio está basado en el ejercicio de la primera versión del videojuego *Phiby´s Adventure* "*Climb the tree*" (Figura 55). En este caso la mecánica de juego de los ejercicios de las dos versiones es básicamente la misma, aunque en el caso de la primera de ellas, no aparecen cohetes durante el ascenso del árbol.

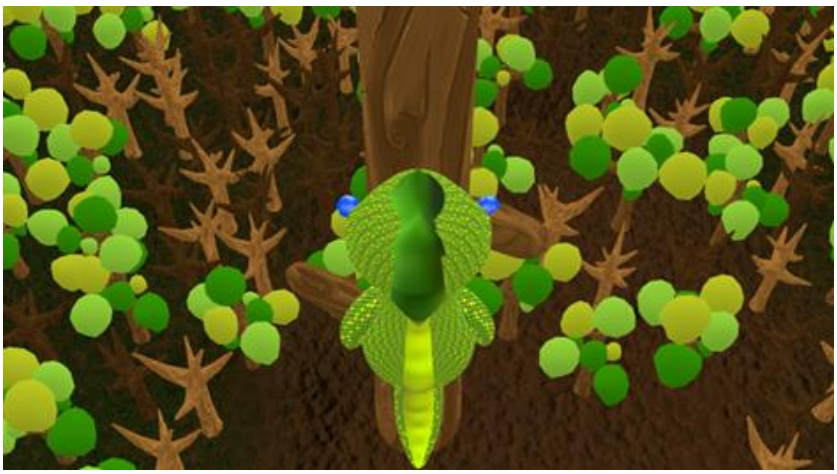


Figura 55. Ejercicio de referencia de la primera version de Phiby´s Adventure para el ejercicio 2 - ¡Trepa el arbol!

- **Objetivo:**

El usuario debe trepar un número de metros indicado a lo largo del tronco de un gran árbol. Para ello debe levantar cada uno de los brazo hasta tocar unas esferas iluminadas junto a la cabeza de *Phiby*, teniendo que alternar entre el uno y el otro. Cada vez que el usuario toca una de las esferas iluminadas asciende un metro por el tronco (Figura 56).

Figura 56. Ventana de inicio del ejercicio 2 - ¡Trepa el arbol!

Este ejercicio se ha diseñado para ejercitar los músculos de los hombros por igual. Está pensado como un ejercicio con muchas repeticiones del movimiento de baja intensidad (Figura 57).

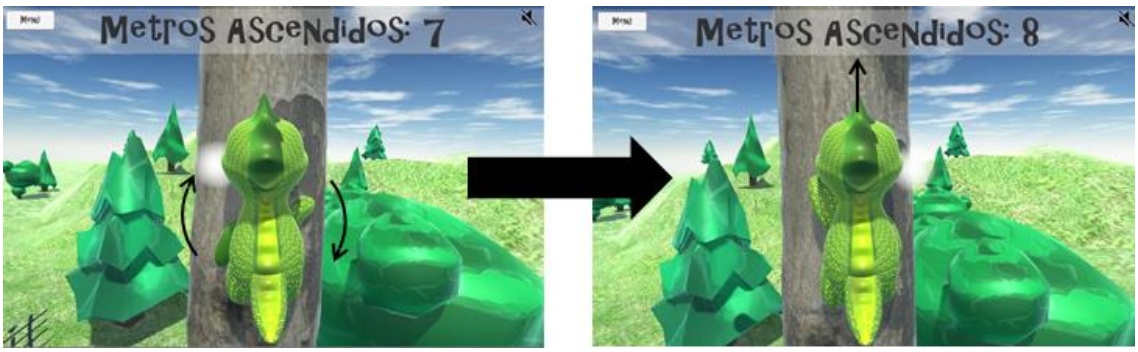


Figura 57. Mecánica de juego del ejercicio 2 - ¡Trepa el arbol!

- **Dificultad:**

En este caso, cada cierto número de metros que *Phiby* ascienda, existe la probabilidad de que se cruce un cohete a su paso, que en caso de que le impactase, haría que descendiese dos de los metros ascendidos. Para evitar este impacto el usuario debe levantar la mirada y comprobar si se aproxima algún cohete antes de realizar su siguiente ascensión (Figura 58).

El terapeuta puede controlar la intensidad del ejercicio a través de la modificación del número de metros que el usuario debe de ascender, la probabilidad de que aparezcan cohetes a su paso y la velocidad a la que se desplazan estos.

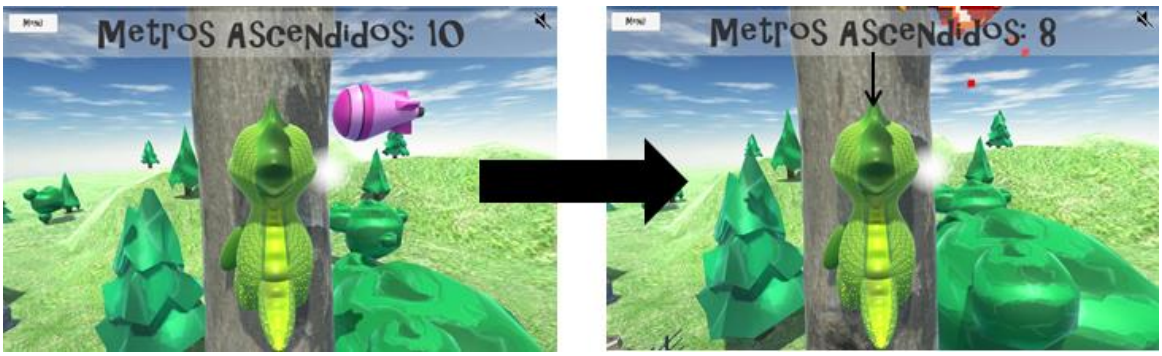


Figura 58. En efecto del impacto de un cohete en el ejercicio 2 - ¡Trepa el arbol!

3.3.3. Ejercicio 3.- ¡Vuelo en ala delta!

Este ejercicio está basado en el ejercicio de la primera versión del videojuego *Phiby´s Adventure* "*Dive and eat*" (Figura 59). En este caso, aunque se ha mantenido el tipo de ejercicio de rehabilitación a realizar, la mecánica de juego ha variado por completo. En la primera versión, el ejercicio se desarrolla en el fondo de un lago por el que el jugador se desplaza libremente y en él debe recoger una serie de ítems que se encuentran dispersos. La dirección hacia la que se realiza el desplazamiento es controlada por el usuario mediante la inclinación que realiza con el torso.

Para esta segunda versión, dado que generar un nuevo entorno acuático fue inviable a nivel temporal, se ideó un ejercicio que, aunque variase por completo el entorno juego, mantuviese las características del ejercicio realizado.

Figura 59. Ejercicio de referencia de la primera version de Phiby´s Adventure para el ejercicio 3 - ¡Vuela en ala delta!

- **Objetivo:**

El usuario debe volar a través de un circuito con una serie de globos que irán apareciendo según toque o sobrepase el globo anterior de la serie. Dado que *Phiby* se desplaza siempre en la dirección a la que apunta la parte delantera del ala delta, el usuario tiene un tiempo limitado para orientar su inclinación y controlar hacia donde se dirige. A modo de ayuda, en la parte superior del modelo hay una flecha de dirección que indica donde se encuentra el próximo globo a recoger (Figura 60).

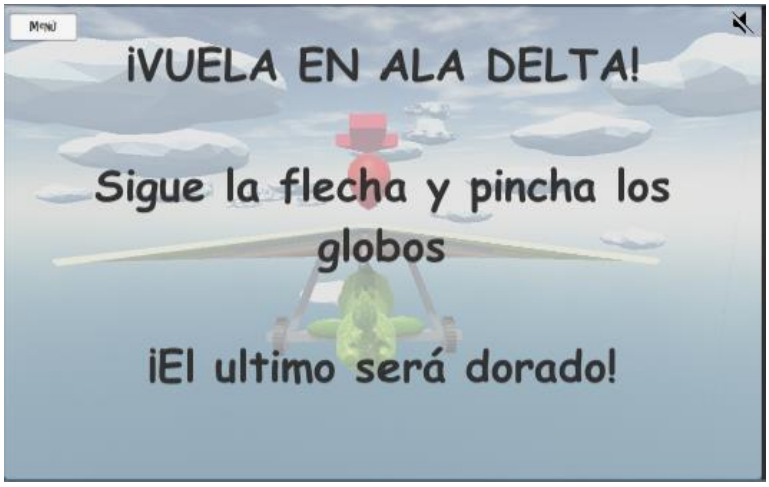


Figura 60. Ventana de inicio del ejercicio 3 - ¡Vuelo en ala delta!

Este ejercicio se ha diseñado para ejercitar la rotación del tronco sobre su propio eje, haciendo al usuario inclinarse frontal y lateralmente. Está pensado como un ejercicio cuya intensidad en la ejecución es muy baja, pero que requiere de una gran movilidad de la articulación (Figura 61).

Figura 61. Mecánica de juego del ejercicio 3 - ¡Vuelo en ala delta!

- **Dificultad:**

En este caso, la dificultad añadida de este ejercicio es que el color del globo que el usuario debe recoger varía cada vez que se genera un nuevo globo. De esta manera, el circuito que el usuario recorre cambia cada vez que realiza este ejercicio. El color del globo que debe ser recogido se indica en la flecha que indica la dirección del globo objetivo (Figura 62).

El terapeuta puede controlar la intensidad del ejercicio a través de la modificación del número de globos que el usuario debe de recoger para finalizar el circuito y la velocidad a la que se desplaza el ala delta.

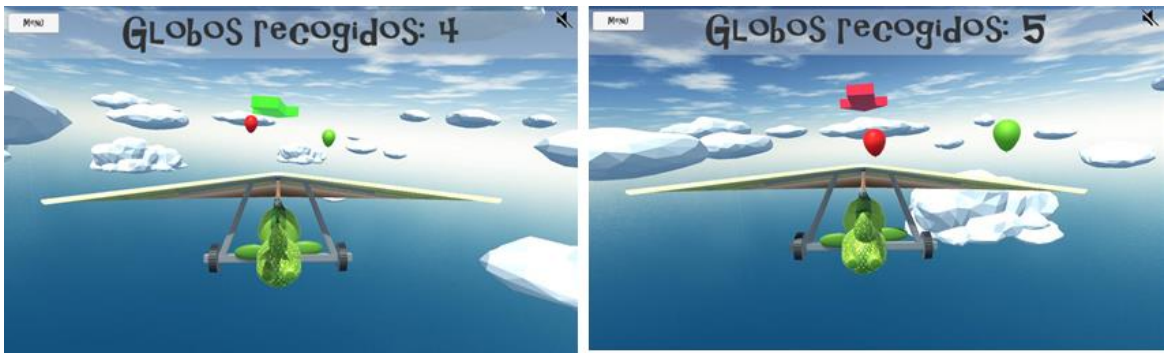


Figura 62. Distintos globos a recoger en el ejercicio 3 - ¡Vuelo en ala delta!

3.3.4. Ejercicio 4.- ¡Navega el rio!

Este ejercicio está basado en el ejercicio de la primera versión del videojuego *Phiby´s Adventure* "*Row the boat*" (Figura 63). En esta segunda versión se ha mantenido el mismo tipo de ejercicio a realizar y la mayor parte de la mecánica del juego, variando únicamente el objetivo final del mismo. Así, en la primera versión el objetivo era recorrer cierta distancia remando con la barca a través de un lago mientras que, en esta segunda, es esquivar las piedras que aparecen durante el recorrido realizado a través de un rio. Con esta modificación, se ha tratado de dar cierto grado de complejidad al ejercicio, de manera que aumente su jugabilidad.

Figura 63 – Ejercicio de referencia de la primera version de Phiby´s Adventure para el ejercicio 4 - ¡Navega el rio!

- **Objetivo:**

El usuario se encuentra en una barca que se desplaza constantemente a través de la corriente de un rio. Mediante el uso de remos puede hacer que esta se desplace lateralmente, realizando para ello el movimiento asociado a la operación de remar. Como en el caso anterior, dado que la barca se desplaza en la dirección de la corriente del rio el usuario tiene un tiempo limitado para decidir a donde se dirige (Figura 64).

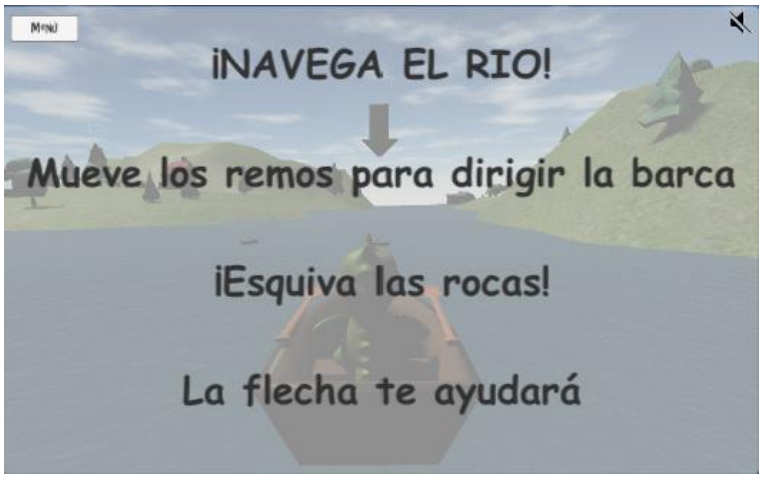


Figura 64. Ventana de inicio del ejercicio 4 - ¡Navega el rio!

Este ejercicio se ha diseñado para ejercitar el movimiento circular de ambos brazos. Está pensado como un ejercicio de resistencia con muchas repeticiones continuas de contracción y estiramiento (Figura 65).

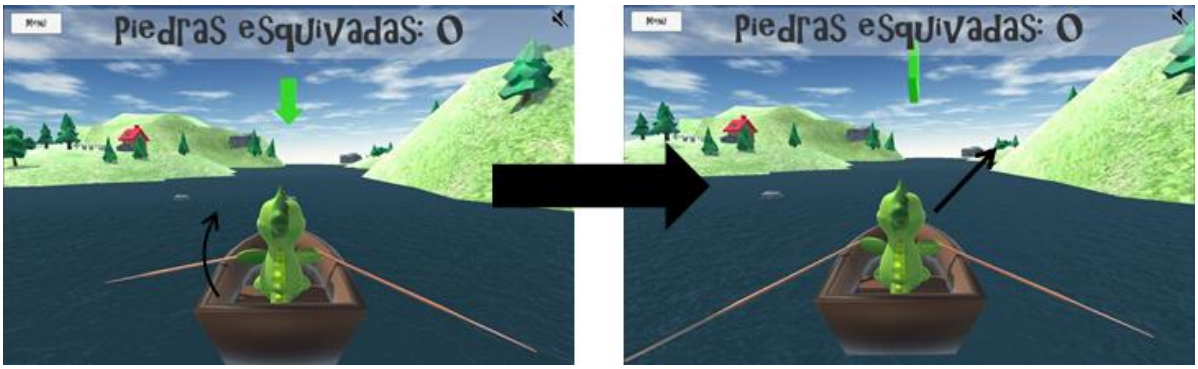


Figura 65. Mecánica de juego del ejercicio 4 - ¡Navega el rio!

- **Dificultad:**

Para este ejercicio la dificultad radica en que, a lo largo de la corriente del rio, aparecen rocas que el usuario debe esquivar remando en dirección contraria a ellas. A nivel de ayuda, se muestra una flecha verde de dirección hacia donde se dirige actualmente la barca, de manera que pueda saber cuándo ha remado lo suficiente para esquivar las rocas (Figura 66).

El terapeuta puede controlar la intensidad del ejercicio a través de la modificación del número de rocas que el usuario debe de esquivar para finalizar el circuito, el número de repeticiones del movimiento que ha de realizar para que la barca se desplace lateralmente y la velocidad a la que se desplaza esta a lo largo del rio.

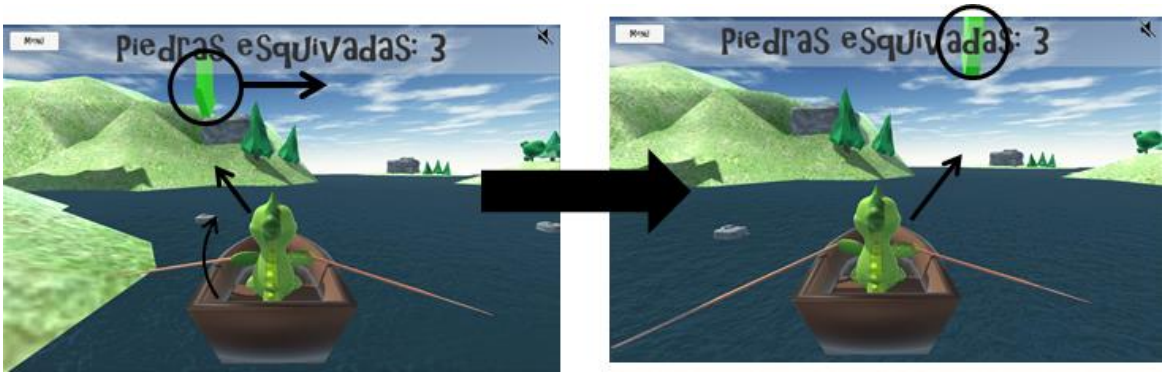


Figura 66. Sistema para la esquiva de las rocas en el ejercicio 4 - ¡Navega el rio!

3.4. Diagrama de bloques final del proyecto

Con estos cuatro ejercicios queda concluida la implementación práctica que demuestra el funcionamiento de las herramientas diseñas en este proyecto. Con su análisis se puede dar por concluido también el apartado referido a la solución tecnológica implementada, habiendo conseguido el objetivo inicial de comunicar Kinect V2 con una aplicación desarrollada en Unity 3D, y que esta procese la información recibida de manera que pueda ser utilizada como elemento de control de un modelo 3D.

Así, a modo de recapitulación, las figuras 67 y 68 muestran dos diagramas de bloques, en los que se engloban la mayor parte de las tareas y operaciones descritas a lo largo del capítulo tercero de este documento. Con ellos se trata de dar al lector una visión estructural completa del diseño utilizado para llevar a cabo la solución tecnológica descrita.

Los bloques mostrados en estos diagramas se dividen en tres categorías que se distinguen mediante el tipo de flecha que tenga asociada:

→ Sucesión de tareas realizadas por cada uno de los hilos secundarios generados tanto por el *middleware* como por Unity 3D. Las tareas están situadas de forma secuencial, de manera, que pueda apreciarse el orden en el que se realizan y la dependencia que existe entre ellas.

→ Mensajes que las aplicaciones intercambian durante su conexión, mostrando su procedencia y el direccionamiento interno que se les aplica.

➡ Los distintos flujos de datos creados por las aplicaciones durante su ejecución, especificando, en cada caso, el tipo de canal de comunicación utilizado.

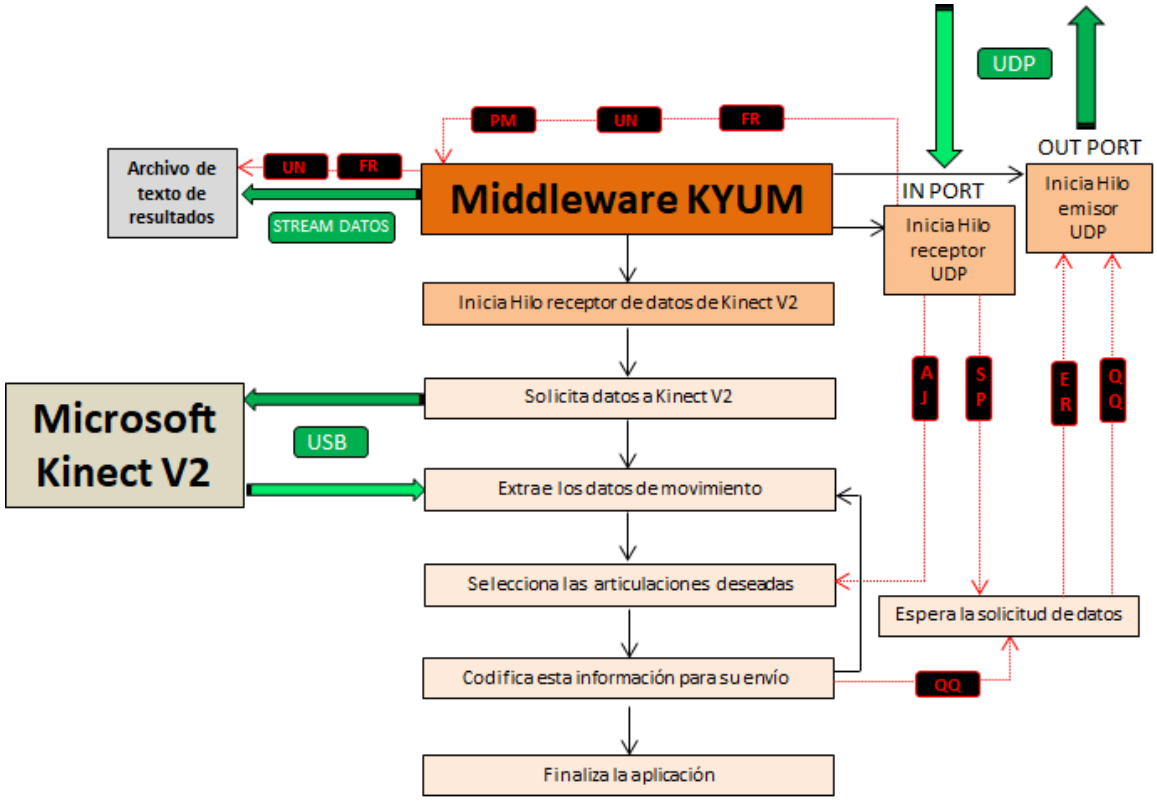


Figura 67. Diagrama de bloques final asociado al middleware K2UM.

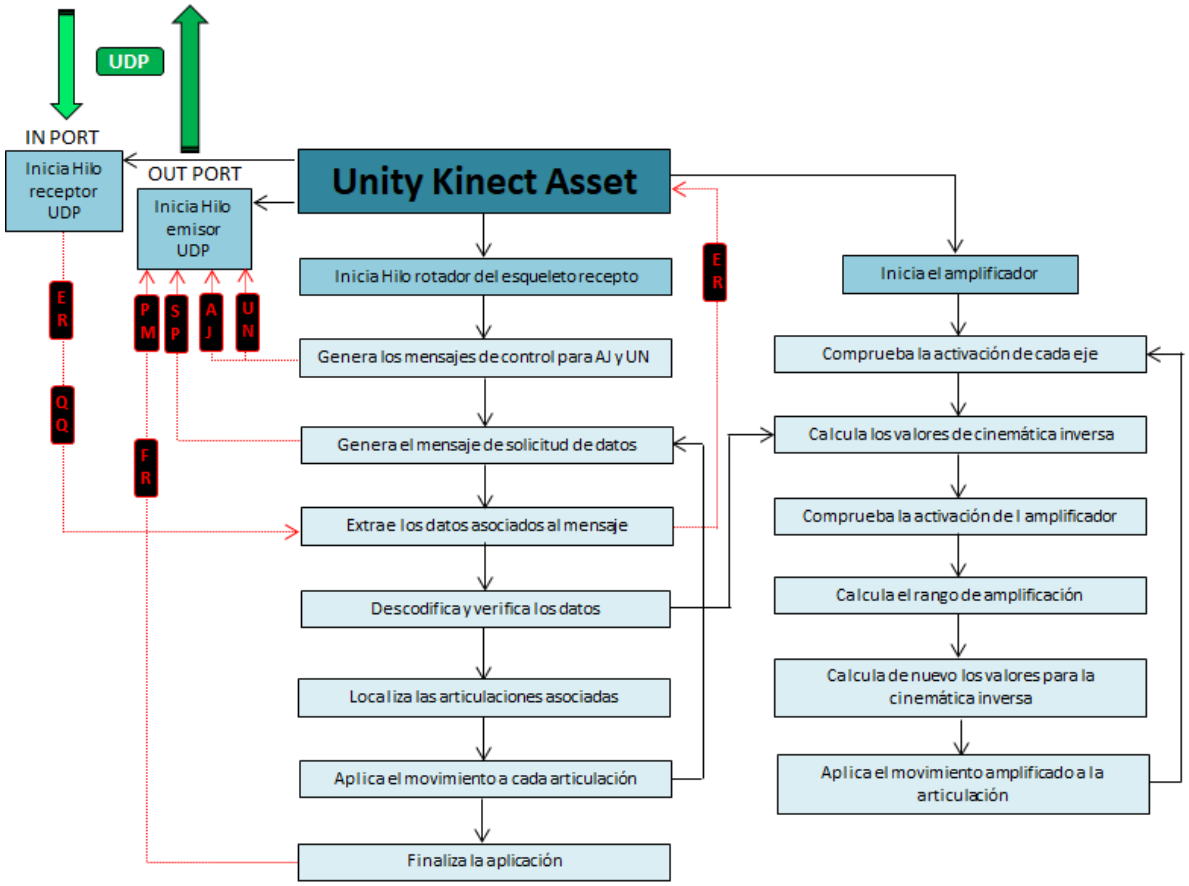


Figura 68. Diagrama de bloques final asociado al asset de Unity 3D Kinect Asset.

Capítulo 4. Resumen de resultados

Dado que el motivo de la realización de este proyecto, es el desarrollo de herramientas software que permitan la comunicación entre el hardware Microsoft Kinect V2 y el software de diseño de videojuegos Unity 3D, el primer paso fue el desarrollo de un *middleware* que permita implementar dicha comunicación. Así, se desarrolla "K2UM", una aplicación que, como función principal, integra un sistema que permite gestionar un canal de comunicación tanto con Kinect V2 como con Unity 3D.

Durante su uso en conjunto con los ejercicios de prueba diseñados, "K2UM" ha demostrado su estabilidad y correcto funcionamiento. Desde ella se ha conseguido establecer un canal de transmisión continuo con el hardware Microsoft Kinect V2 y recibir con fluidez los datos que este transmite. También se ha logrado seleccionar, codificar y enviar vía UDP los datos asociados al movimiento transmitidos por el hardware, permitiendo el envío, tanto de la posición espacial, como de la rotación asociada a cada articulación. Por último, también se ha logrado implementar de forma correcta un interface gráfico que permita su fácil acceso al usuario y la tarea de almacenar los resultados de los ejercicios que este realiza.

Mediante la descarga del modelo 3D de un esqueleto humano, se pudo comprobar tanto la funcionalidad del esqueleto receptor de Unity 3D, como del amplificador de movimientos diseñado. En el caso del esqueleto receptor, los resultados fueron los esperados consiguiendo, que el modelo 3D asociado reprodujera con mucha exactitud los movimientos captados por el hardware Microsoft Kinect V2. En el caso del amplificador de movimientos, aunque los resultados obtenidos en su uso en conjunto con el modelo 3D son los buscados, al conseguir una amplificación optima de la articulación seleccionada, limita en cierto modo el grado de movimiento de dicha articulación en el modelo 3D, de manera que, en el algunos casos, esta no llega a su máxima rotación posible por algunos grados. Sin embargo, dado que en el modelo 3D diseñado para el proyecto *BLEXER* está limitación no es apreciable en los movimientos que este realiza, y esta herramienta en concreto está diseñada para su uso en conjunto con este modelo 3D, no es una incidencia que se considere prioritaria a solucionar en el uso futuro de esta herramienta.

Por último, la ejecución de la aplicación asociada a los cuatro ejercicios de prueba ha demostrado también como estas herramientas software diseñadas pueden ser utilizadas para el desarrollo de una nueva versión del videojuego "*Phiby´s Adventure*" implementada en el software de diseño Unity 3D, permitiendo el uso de los modelos 3D y escenarios ya diseñados para la versión anterior de este videojuego. También han intentado mostrar la potencia del motor para videojuegos que implementa el software Unity 3D, y que justifica ampliamente su uso en sustitución del software utilizado para el diseño de las anteriores versiones de este videojuego.

Capítulo 5. Conclusiones y posibles líneas futuras de trabajo

5.1. Conclusiones

En la introducción mostrada en el capítulo 1 de este documento se listan una serie de objetivos a realizar durante la ejecución de este proyecto. El primero de ellos es generar mediante un *middleware*, un canal de comunicación estable entre el hardware Microst Kinect V2 y el entorno de desarrollo Unity 3D.Este objetivo se puede dar como logrado al conseguirse que el *middleware* "K2UM" gestione con fluidez y sin fallos esta comunicación.

El siguiente objetivo es el de procesar los datos procedentes del *middleware* en Unity 3D, de manera que, se utilicen para controlar los movimientos de un personaje. Con el complemento diseñado para Unity 3D, `KinnectAssetUnity,` se logra llevar a cabo este objetivo, consiguiendo que los movimientos transferidos por el *middleware*, sean aplicados con exactitud a un modelo 3D. También se ha conseguido generar un amplificador de movimientos permite simular rotaciones completas en las articulaciones del modelo 3D cuando el usuario captado tiene cierto grado de movilidad reducida.

Otro de los objetivos presentados es el de la simplicidad en el acceso tanto al *middleware* como a los ejecutables que contienen los ejercicios de rehabilitación a realizar. El interfaz gráfico del *middleware* se diseña para cumplir este objetivo, permitiendo su activación con una única pulsación sobre uno de sus controles. En el caso de la versión de demostración realizada, este objetivo también se cumple, ya que, los cuatro ejercicios diseñados están asociados a un menú de carga que permite al usuario acceder a ellos también con solo una pulsación.

Los dos últimos objetivos están asociados al desarrollo de los distintos ejercicios de rehabilitación que componen la aplicación y del entorno virtual en el que se encuentran estos. En este caso, estos objetivos solo han podido ser cumplidos en parte. Así, las limitaciones temporales solo han permitido crear cuatro de los ejercicios de rehabilitación inicialmente ideados, y un entorno virtual muy limitado en el que estos se sitúan.

5.2. Posibles futuras líneas de trabajo

Además de los objetivos comentados en el apartado anterior, existen otras necesidades que, aunque inicialmente no eran imprescindibles para esta primera versión desarrollada, si serán necesarias en fases futuras del proyecto. Estas pueden ser las líneas de trabajo futuras más probables, y la más importante de ellas es la comunicación con la página WEB donde el terapeuta puede interactuar con el *middleware* y acceder a los resultados de los ejercicios realizados por los usuarios.

Durante el desarrollo de este proyecto, aunque no se ha llegado a implementar esta comunicación WEB, sí se ha tenido en cuenta cuando se diseñó el sistema de transmisión entre el *middleware* y Unity 3D. Así, los resultados y nombre del usuario que los ha obtenido, son almacenados por el *middleware* en un archivo de texto, con lo que, en un futuro, solo sería necesario implementar el algoritmo de comunicación con la página WEB que se encargue de realizar el envío de estos resultados.

Otra de las necesidades que deben de ser tenidas en cuenta para el futuro del proyecto es la que se relaciona con la desaparición del mercado del hardware Microsoft Kinect V2. Así, sería necesario realizar una nueva investigación para conocer que nuevos dispositivos de captura de movimiento existen en el mercado y si alguno de ellos se adapta a las necesidades del proyecto. Dado que el *middleware* se ha diseñado internamente de manera modular, solo habría que añadirle el código para comunicar con este nuevo hardware, pudiéndose mantener todo el sistema de comunicación con Unity 3D.

Otra línea de trabajo probable es completar los objetivos comentados anteriormente que no han podido llevarse a cabo por completo. Aunque, debido a limitaciones de temporales, solamente han podido ser implementados los cuatro ejercicios clásicos que ya existían en la versión anterior de "*Phiby´s Adventure*", la solución tecnológica inicialmente diseñada, en conjunto con el desarrollador original de la aplicación Ignacio Gómez-Martinho González, incluía una versión funcional del videojuego mucho más extensa.

Este videojuego consistiría en un escenario de tamaño considerable en el cual el jugador puede desplazarse con relativa libertad y en el que se encuentran dispersos, además de modelos 3D que dan vida y dinamismo a escenario, los distintos ejercicios que puede realizar. Estos ejercicios se mostrarían en forma de eventos de juego ligados a objetos interactuables relacionados con el ejercicio a realizar y que, en algunos casos, permiten el acceso a nuevas zonas del escenario.

El objetivo de este tipo de realización del escenario es que los ejercicios de rehabilitación formen parte del entorno de juego, de manera que, no sean tomados por el usuario como ejercicios independientes, si no como parte de un mundo vivo que puede recorrer. Este concepto se puede utilizar como base para mejorar la experiencia de juego del usuario de manera que haga al videojuego lo más atractivo posible. Esto se puede realizar mediante, por ejemplo, la modificación de zonas del escenario cuando el usuario acceda a ellas, un sistema de recursos que permita al usuario la construcción de nuevos elementos jugables o la aparición de personajes que interactúen con el usuario y formen parte de los ejercicios realizados.

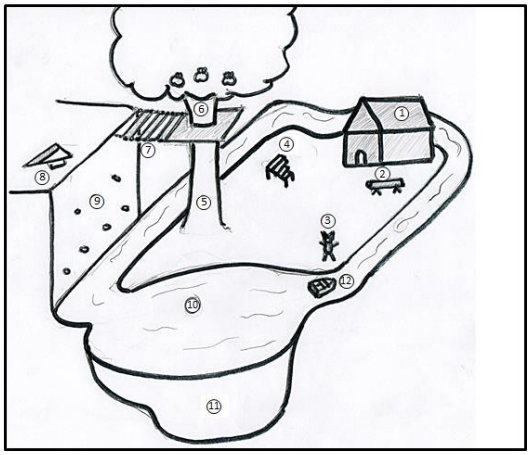


Figura 69 – Posible mapa de escenario para el juego completo Phiby´s Adventure.

La figura 69 muestra una posibilidad para el mapa del escenario. En él jugador podría desplazarse, recorrerlo, e interactuar con los objetos que tienen asociados los ejercicios para iniciarlos. Los números que aparecen en este mapa se relacionan con los posibles ejercicios que podrían añadirse a versiones futuras del videojuego.

A continuación se muestra una lista con una serie de posibles ejercicios que pueden ser incluidos en el mapa de la figura 69:

I. Casa de la abuelita.

No es un ejercicio como tal, sino un espacio donde el jugador puede acudir y revisar los progresos realizados en cada ejercicio.

II. ¡Corta el tronco!

El jugador debe de cortar una serie de troncos mediante la elevación de uno de los brazos hasta una altura y su posterior descenso. Este es uno de los ejercicios implementados en la versión de prueba ejecutable.

III. El imitador de posturas.

El jugador debe de imitar los movimientos de otro personaje representado mediante un modelo 3D de un animal con aspecto humanizado. La aplicación mide el índice de coincidencia entre los movimientos realizados por el usuario y los realizados por el modelo 3D del animal.

IV. ¡Música maestro!

El jugador se encuentra con un instrumento similar a un xilófono sobre el aparecen flechas para indicar que teclas deben de ser tocadas. Cada una de estas teclas está asociada a un sonido en concreto, de manera que, mientras suena una melodía, se realicen cortes en ella, y se indique al usuario que golpee con un mazo sobre la tecla con el sonido asociado a la melodía en ese instante.

V. ¡Trepa el tronco!

El jugador debe escalar hasta lo alto de un tronco mientras esquiva cohetes que se encuentra en su camino. Para realizar el ascenso el jugador debe de elevar ambos brazos alternando entre uno y otro. Este es uno de los ejercicios implementados en la versión de prueba ejecutable. Este ejercicio da acceso a la zona donde se encuentran los ejercicios 6 y 7.

VI. ¡Recoge las manzanas!

El jugador debe recoger en un cesto las manzanas que caen desde lo alto de un árbol. Para ello debe desplazar sus manos hacia la izquierda, la derecha y por encima de su cabeza para decidir donde se encuentra la cesta. Algunas de las manzanas pueden estar podridas y no deben de ser recogidas por el usuario.

VII. ¡Cruza el puente colgante!

El jugador debe recorrer un puente colgante inestable que se inclina lateralmente en la dirección del viento que sopla. Cuando esto suceda, la aplicación activa un indicador que informa al jugador de que debe de inclinar su tronco hacia el lado contrario. Este ejercicio da acceso a la zona donde se encuentra el ejercicio 8.

VIII. Vuelo en ala delta.

El usuario debe recorrer un camino a través de un circuito con una serie de globos que irán apareciendo según el usuario toque o sobrepase el globo anterior de la serie. El usuario controla la dirección del ala delta mediante la inclinación del tronco. También debe de elegir entre dos globos, verde y rojo respectivamente, en función del color objetivo que se le indique. Este es uno de los ejercicios implementados en la versión de prueba ejecutable.

IX. ¡Escala el muro!

El jugador debe escalar un muro mediante el desplazamiento de los brazos hasta presas que aparecen iluminadas en el muro, ascendiendo unos metros cada vez que toca una. El jugador también debe de esquivar piedras que caen desde lo alto del muro tomando distintos caminos durante la escalada. Este ejercicio da acceso a la zona donde se encuentra el ejercicio 8.

X. ¡Navega el lago!

El jugador debe remar en una barca hasta un punto del lago mediante la realización del movimiento asociado a la operación de remar. El punto al que debe dirigirse cambia con cada realización del ejercicio, de manera que el usuario debe de controlar la dirección en la que avanza. Este ejercicio da acceso a la zona donde se encuentra el ejercicio 11.

XI. La búsqueda del tesoro.

El jugador debe sumergirse en el lago y bucear hasta la posición muestra un indicador en la que se encuentra un tesoro sumergido. El jugador controla la dirección mediante la inclinación del tronco y avanza realizando el movimiento asociado a la operación de nadar bajo el agua.

XII. ¡Navega el rio!

El jugador debe controlar la dirección en la que se desplaza una barca mientras es arrastrada por la corriente de un rio. Para ello debe realizar el movimiento asociado a la operación de remar consiguiendo que la barca se desplace lateralmente en la corriente del rio. Este es uno de los ejercicios implementados en la versión de prueba ejecutable.

Referencias

[1] UPM «Grupo de Aplicaciones Multimedia y Acústica (GAMMA) ». *Fecha de última consulta 02/02/2018*
https://www.citsem.upm.es/index.php/es/personal/grupos/personal-gamma

[2] UPM «Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM)». *Fecha de última consulta 02/02/2018*
https://www.citsem.upm.es

[3] Microsoft «Kinect - Desarrollo de Aplicaciones» *Fecha de última consulta 02/02/2018*
https://developer.microsoft.com/es-es/windows/kinect

[4] Ignacio Gómez-Martinho González, «Desarrollo e implementación de middleware entre Blender, Kinect y otros dispositivos» Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, UPM, Madrid, Julio 2016.

[5] Microsoft «How Microsoft and Novartis created Assess MS» *Fecha de última consulta 02/02/2018*
https://news.microsoft.com/features/from-gaming-system-to-medical-breakthrough-how-microsoft-and-novartis-created-assess-ms/t

[6] Precission Games «Human Motion» *Fecha de última consulta 02/02/2018*
http://www.precission.com.ar/portfolio-items/human-motion-games/

[7] Virtual Ware Group «VirtualRehab» *Fecha de última consulta 02/02/2018*
http://virtualwaregroup.com/es/productos/virtualrehab

[8] Blender Proyect *Fecha de última consulta 02/02/2018*
https://www.blender.org/

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

[10] Unity Technologies *Fecha de última consulta 02/02/2018*
https://unity3d.com/es/

[11] Computer Hoy «Así es Kinect V2» *Fecha de última consulta 02/02/201,*
https://computerhoy.com/noticias/hardware/asi-es-kinect-20-windows-pc-10937

[12] Academia Android «Motor de juegos Unity 3D» *Fecha de última consulta 02/02/201*, https://academiaandroid.com/motor-de-juegos-unity-3d/

[13] Microsoft «Kinect for Windows SDK 2.0» *Fecha de última consulta 02/02/2018* https://www.microsoft.com/en-us/download/details.aspx?id=44561

[14] Microsoft «Kinect for Windows v2 Windows Runtime API Reference» *Fecha de última consulta 02/02/2018* https://msdn.microsoft.com/en-us/library/dn758675.aspx

[15] Microsoft «JointType Enumeration». *Fecha de última consulta 02/02/2018* https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

[16] 3dgep «Understanding Quaternions» *Fecha de última consulta 02/02/2018* https://www.3dgep.com/understanding-quaternions/

[17] StackOverflow «C# UDP Socket client and server» *Fecha de última consulta 02/02/2018* https://stackoverflow.com/questions/19786668/c-sharp-udp-socket-client-and-server

[18] Matthias Kronlachner «Kinect Headtracking with OSC Support» *Fecha de última consulta 02/02/2018* http://www.matthiaskronlachner.com/?p=624

[19] WireShark Software *Fecha de última consulta 02/02/2018* www.wireshark.org

[20] Unity Technologies «Asset Workflow» *Fecha de última consulta 02/02/2018* https://docs.unity3d.com/es/current/Manual/AssetWorkflow.html

[21] Unity Technologies «Asset Packages » *Fecha de última consulta 02/02/2018* https://docs.unity3d.com/Manual/AssetPackages.html

[22] Unity Technologies «Prefabs» *Fecha de última consulta 02/02/2018* https://docs.unity3d.com/es/current/Manual/Prefabs.html

[23] Unity Technologies «GameObjects» *Fecha de última consulta 02/02/2018* https://docs.unity3d.com/es/current/Manual/GameObjects.html

[24] DarkCom «Desmitificando las corrutinas» *Fecha de última consulta 02/02/2018* https://darkcom.wixsite.com/darkcomtech/single-post/2017/06/18/Desmitificando-las-corrutinas

[25] Microsoft «How to interpret JointOrientation data» *Fecha de última consulta 02/02/2018* https://social.msdn.microsoft.com/Forums/en-US/a87049b5-7842-4c17-b776-3f6f4260c801/how-to-interpret-jointorientation-data?forum=k4wv2devpreview

[26] Unity Technologies «AssetStore» *Fecha de última consulta 02/02/2018*
https://assetstore.unity.com/

[27] Unity Technologies «Using Inspector» *Fecha de última consulta 02/02/2018*
https://docs.unity3d.com/es/current/Manual/UsingTheInspector.html

[28] Research Gate «Fordward and Inverse Kinematic» *Fecha de última consulta 02/02/2018*
https://www.researchgate.net/post/What_is_the_difference_between_forward_kinematics_and_inverse_kinematics

[29] Unity Technologies « Questions and Answers» *Fecha de última consulta 02/02/2018*
https://answers.unity.com/questions/11106/how-to-do-inverse-kinematics-ik-in-unity.html

Anexo I. Cómo utilizar la aplicación `PhibysDemo.exe` en conjunto con el *middleware* `K2UM.exe`

Como muestra funcional de las herramientas software diseñadas en este proyecto se ha desarrollado un ejecutable con cuatro ejercicios de prueba basados en la anterior versión de este videojuego. A continuación se muestran los pasos a seguir para el correcto acceso al videojuego.

I. Iniciar el *middleware* accediendo a la carpeta "*Middleware K2UM*" y ejecutando el archivo **`K2UM.exe`.**

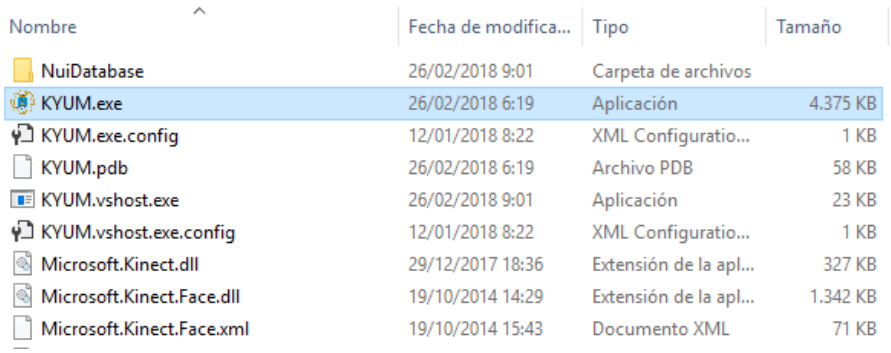


Figura 70. Paso 1 del uso conjunto de la aplicación "PhibysDemo.exe" y el middleware "K2UM.exe"

II. Una vez iniciado el *middleware* pulsar sobre el botón asociado al texto "*Iniciar aplicación*".

Figura 71. Paso 2 del uso conjunto de la aplicación "PhibysDemo.exe" y el middleware "K2UM.exe"

III. Iniciar el videojuego accediendo a la carpeta "*Phibys Adventure Unity3D Demo Version"* y ejecutar el archivo **`PhibysDemo.exe`**.

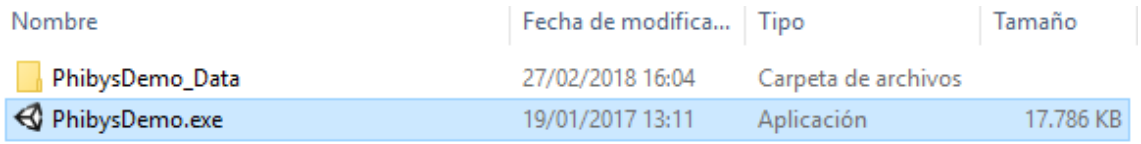


Figura 72. Paso 3 del uso conjunto de la aplicación "PhibysDemo.exe" y el middleware "K2UM.exe"

Anexo II. Cómo realizar la configuración completa del asset *KinectAssetUnity*

El asset *KinectAsset* para Unity 3D, incluye dos objeto que permiten implementar el movimiento transmitido por el *middleware* en un modelo 3D y la amplificación de una articulación de este modelo 3D. De la correcta configuración de este *asset* depende el grado de fidelidad en el movimiento que va a realizar el modelo 3D y la articulación amplificada, por ello, a continuación se muestran los pasos a seguir para realizar de forma óptima esta operación.

I. Importar el modelo 3D a utilizar arrastrándolo a la carpeta de *asset* del proyecto y posicionarlo en la escena.

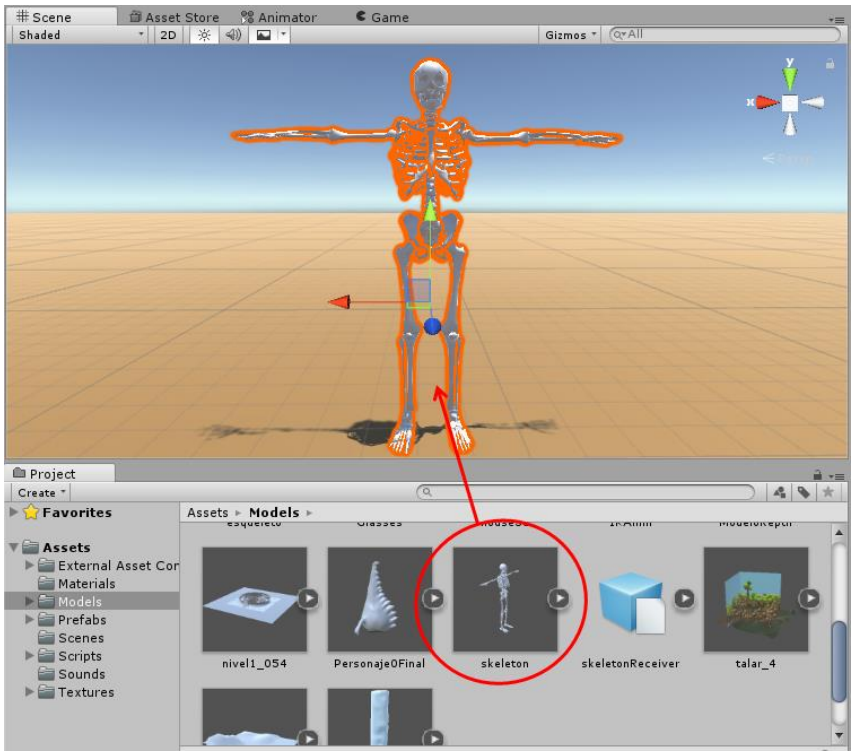


Figura 73. Paso 1 de la configuracion del objecto AmplifyManager

II. Importar el asset KinectAsset seleccionando en la barra de herramientas la opción "*Assets*", y dentro de ella "*Import Package*" – "Custom Package".

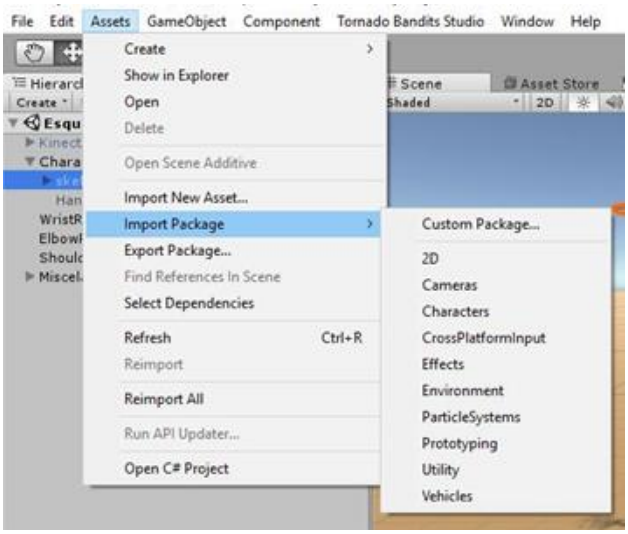


Figura 74. Paso 2 de la configuracion del objecto AmplifyManager

III. Arrastrar los objetos *KinectReceiver* y *AmplifyManager* incluidos en la carpeta "*Prefabs*".

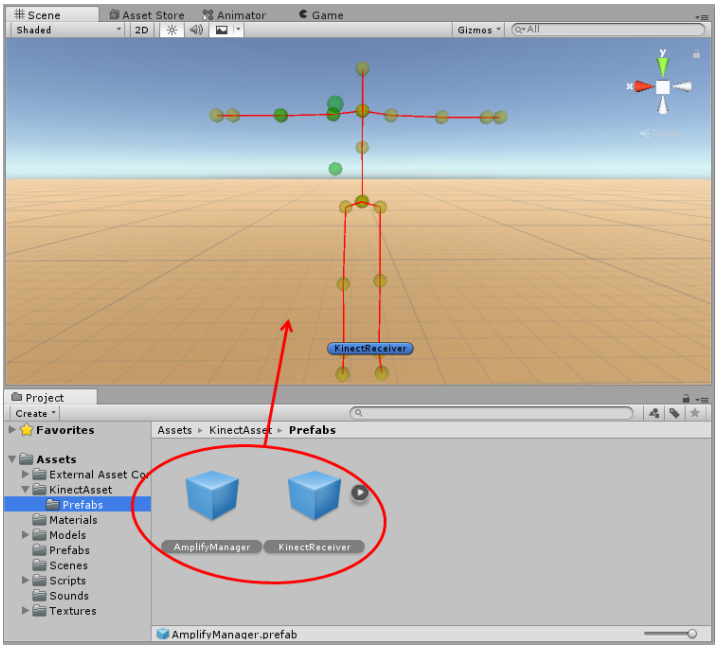


Figura 75. Paso 3 de la configuracion del objecto AmplifyManager

IV. Posicionar objeto *SkeletonReceiver*, emparentado con el objeto *KinectReceiver*, en el interior del modelo 3D, haciendo coincidir, con la mayor exactitud posible, la posición de las articulaciones del objeto *SkeletonReceiver* con las del modelo 3D.

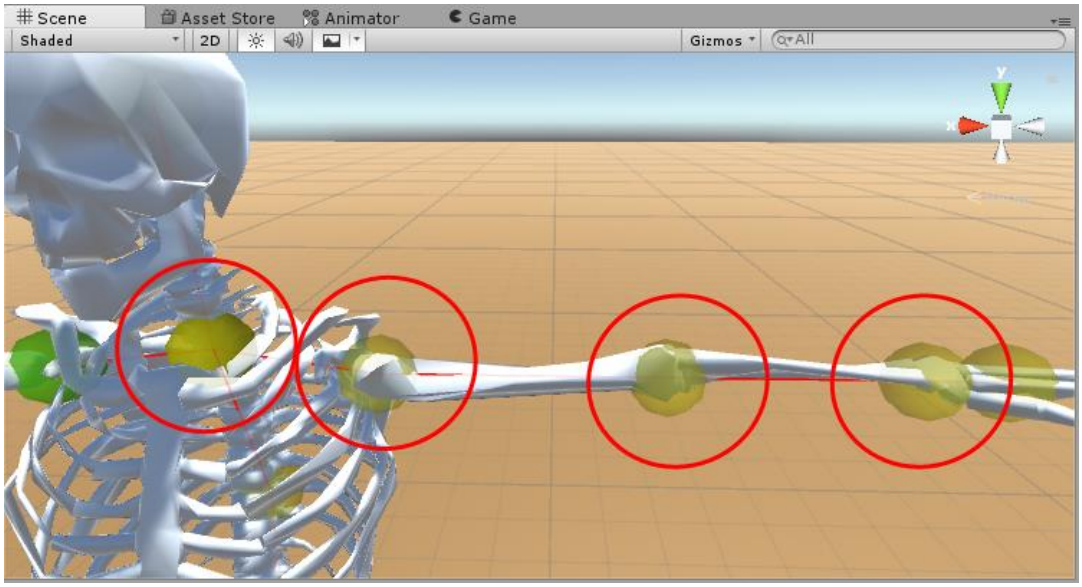


Figura 76. Paso 4 de la configuracion del objecto AmplifyManager

V. Asignar cada articulación del modelo 3D a su equivalente en del objeto *SkeletonReceiver* a través del objeto *KinectReceiver*.

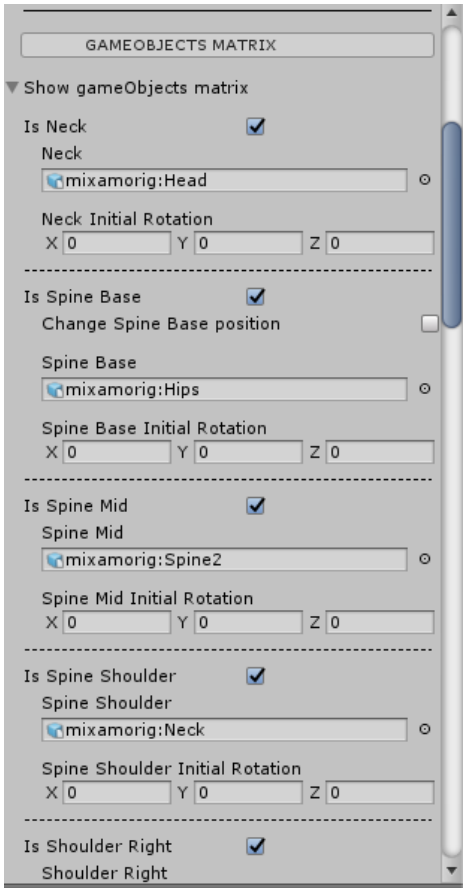


Figura 77. Paso 5 de la configuracion del objecto AmplifyManager

Para esta prueba únicamente se va a realizar la amplificación del eje vertical aunque la cinemática inversa estará activada en ambos ejes.

VI. Situar los calibradores incluidos en la carpeta "*Calibrators*" emparentada con el objeto *AmplifyManager*. El calibrador *CalibratorJointEdge* se sitúa en la articulación a amplificar. El calibrador *CalibratorEndOfBone* se sitúa en la siguiente articulación del eje jerárquico. Los otros dos calibradores *CalibratorVerticalOrigin* y *CalibratorHorizontalOrigin* se sitúan en la posición que desee ser tomada como punto de origen del movimiento que realiza para cada uno de los ejes el objeto *CalibratorEndOfBone*.

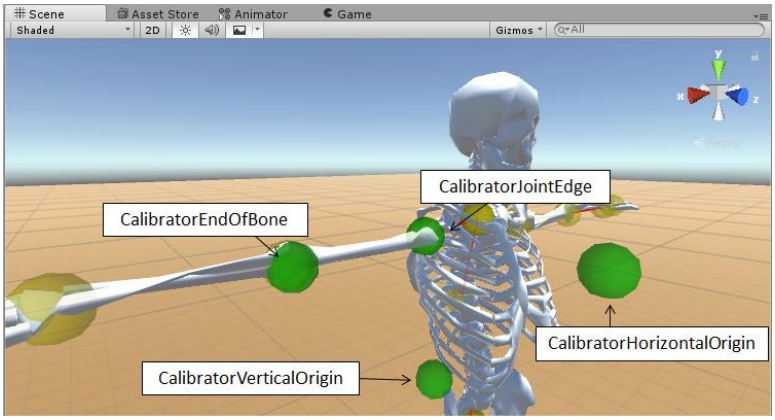


Figura 78. Paso 6 de la configuracion del objecto AmplifyManager

VII. Asociar la articulación a amplificar del objeto *SkeletonReceiver* en el objeto *AmplifyManager* y activar la cinemática inversa del eje vertical y el modo calibrado.

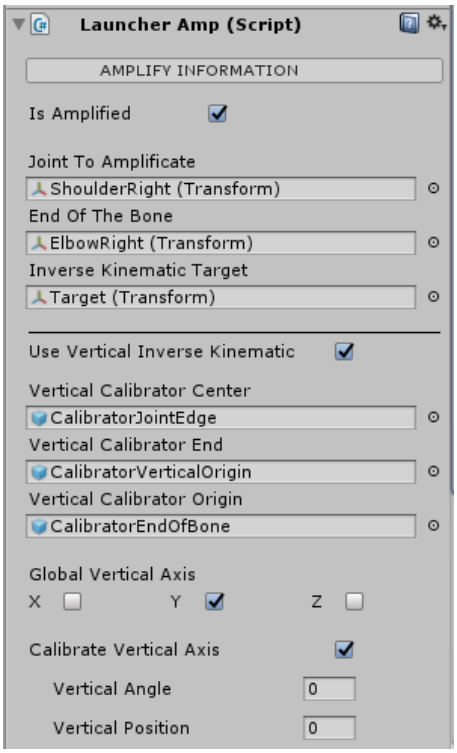


Figura 79. Paso 7 de la configuracion del objecto AmplifyManager

VIII. Situar el *GameObject* "*Target*" emparentado con el *prefab AmplifyManager* en el extremo del hueso ligado a la articulación a amplificar.

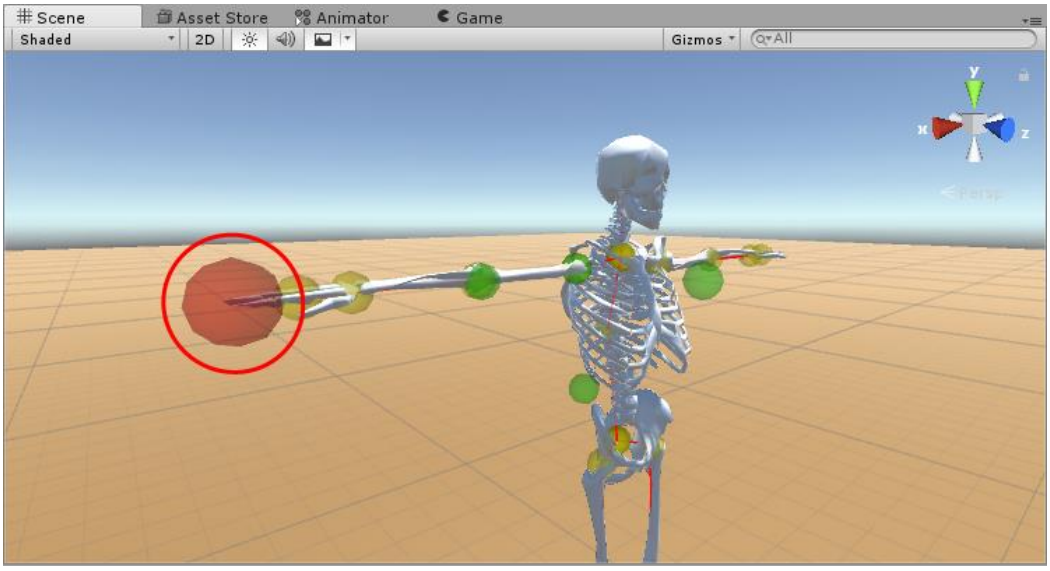


Figura 80. Paso 8 de la configuracion del objecto AmplifyManager

IX. Iniciar la ejecución de la aplicación dentro de Unity 3D y desplazar el *GameObject Target* para localizar las posiciones de ese objeto *Target* en las que la rotación de la articulación a amplificar es máxima y mínima.

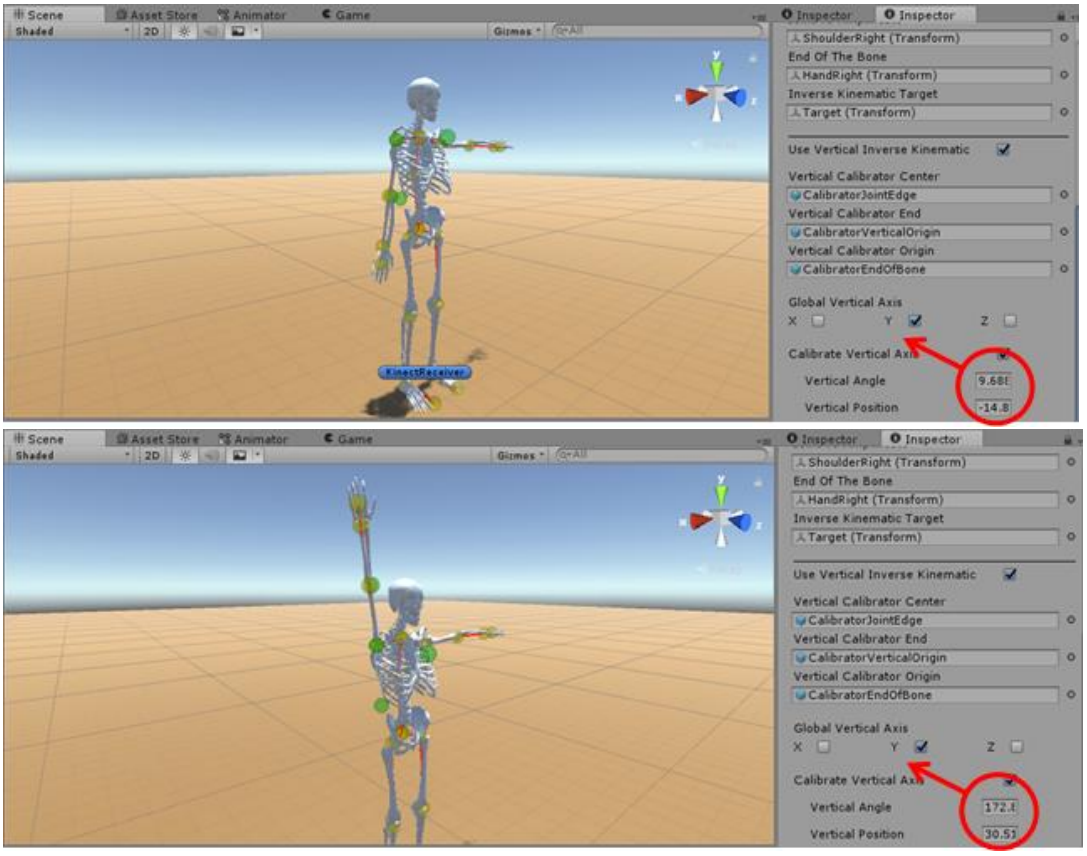


Figura 81. Paso 9 de la configuracion del objecto AmplifyManager

X. Introducir los datos medidos en la información de la cinemática inversa del eje y activar su amplificación.

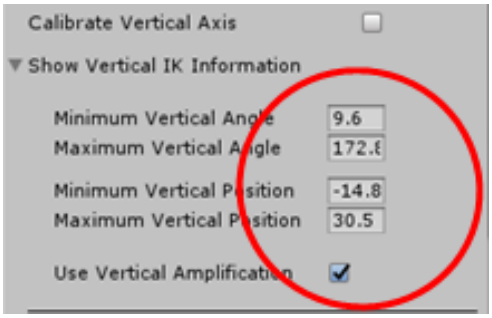


Figura 82. Paso 10 de la configuracion del objecto AmplifyManager

XI. Repetir la operación en el eje horizontal pero sin activar la amplificación.

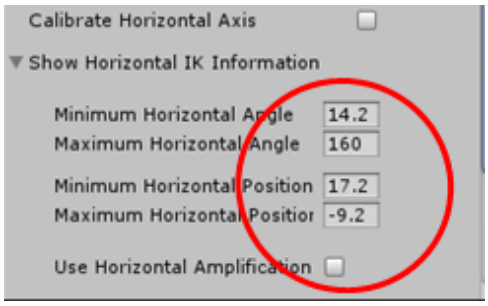


Figura 83. Paso 11 de la configuracion del objecto AmplifyManager

XII. Una vez calibrado, asociar la articulación a amplificar al objeto *AmplifyManager*, pero en este caso en vez de asociar la que pertenece al objeto *SkeletonReceiver* se asocia la perteneciente al modelo 3D.

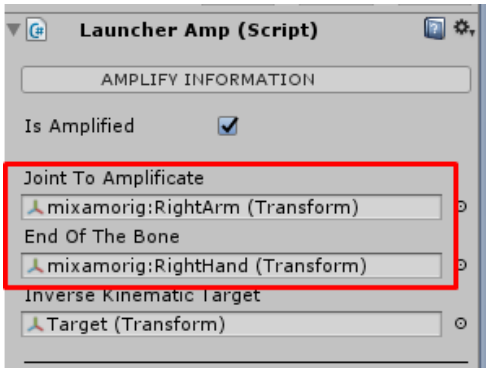


Figura 84. Paso 12 de la configuracion del objecto AmplifyManager

XIII. Por último se asocian *GameObjects* vacíos a las articulaciones implicadas en la amplificación dentro del objeto *KinectReceiver*, para que únicamente actúe el objeto *AmplifyManager* en la rotación de la articulación a amplificar del modelo 3D.

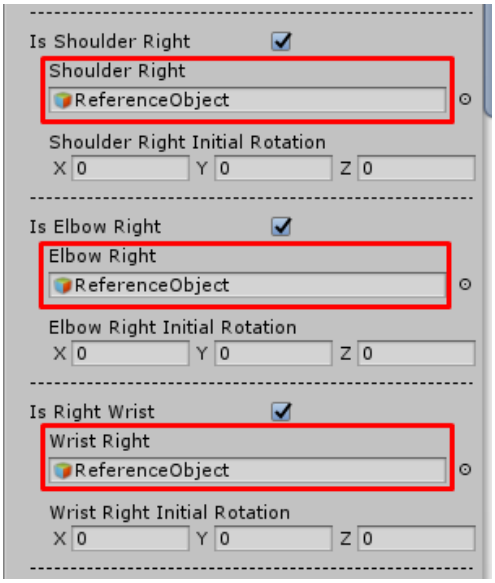


Figura 85. Paso 13 de la configuracion del objecto AmplifyManager

Anexo III. Código fuente

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

Anexo IV. Presupuesto

A continuación se muestra un resumen tanto de los costes económicos asociados al proyecto, como de las horas de trabajo utilizadas para llevar a cabo cada una de sus partes.

Materiales:

- Equipo Informático……...480 €
  - Intel® Core 2 6400 a 2.13 GHz
  - 4 GB RAM DDR3
  - 500 GB Disco Duro 5400 rpm
  - Intel Graphics
- Microsoft Kinect V 2.0…….…...140 €
- Adaptador Microsoft Kinect V2 para PC compatible ....…….……20 €

Costes totales de materiales……..640€

Horas de trabajo:

- 6 meses de trabajo realizando prácticas de colaboración en CITSEM
  - 25 horas semanales ……..600 horas totales
- 4 meses de mejora y ampliación del trabajo desarrollado en las prácticas
  - 25 horas semanales ……400 horas totales
- 2 meses de comprobación de resultados y redacción del informe
  - 20 horas semanales ……160 horas totales

Horas TOTALES……...……..1060 horas totales

Precio hora ingeniero……...……20 €/h

Costes de las horas de trabajo…….……..21200€

COSTE TOTAL DEL PROYECTO…….21840€