MEMORIA DE PRÁCTICA EXTERNA
ETS DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN
UPM

Andrés Ramírez González

1. INTRODUCCIÓN:

Las prácticas realizadas se han basado en la continuación del desarrollo de un videojuego terapéutico dirigido a niños y jóvenes con discapacidad física mediante el cual se pretende ayudar a los usuarios a mejorar y entrenar sus capacidades motrices a partir de la realización de ejercicio físico adaptado a sus necesidades. El objetivo final de esta aplicación es ofrecer una herramienta, tanto a terapeutas como a pacientes, para llevar a cabo un tratamiento de manera eficaz sin que los ejercicios a realizar se conviertan en una obligación. Así, el videojuego sobre el que se ha trabajado se denomina "Phiby's Adventure", cuya idea principal radica en la creación de un mundo abierto en el cual se desarrolla una historia o narrativa a la cual el personaje principal debe hacer frente. De esta manera, a Phiby, es decir al jugador, se le propone la realización de determinados ejercicios en forma de minijuegos dentro de un contexto atractivo, cada uno de ellos con unas motivaciones y objetivos claros que aportan interés a los mismos y que están enfocados a que la historia del juego avance de manera interesante a la vez que el paciente realiza el mencionado ejercicio físico fijado en su tratamiento personalizado. En este sentido, todos los ejercicios son configurables a distancia por el terapeuta mediante parámetros específicos e independientes a través de la web médica Blexer-med, de manera que se adapten, como se ha dicho, a las necesidades de cada paciente.

En cuanto a las tecnologías utilizadas para desarrollar el videojuego se han de destacar Kinect, Unity y K2UM (Kinect to Unity Middleware), todas ellas facilitadas por el Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM), el cual se creó en 2011 con el fin de fortalecer y promover la I+D+i y la captación de talento investigador en el Campus Sur de la UPM. Actualmente cuenta con tres grupos de investigación reconocidos por la UPM: Grupo de Tecnología Software y Sistemas (SYST), Grupo de Diseño Electrónico y Microelectrónico (GDEM) y Grupo de Aplicaciones MultiMedia y Acústica (GAMMA), al cual pertenece el laboratorio en el que se han realizado las prácticas y cuya línea principal de investigación abarca las interfaces naturales para personas con discapacidad física mediante videojuegos serios.

El trabajo aportado al videojuego ha consistido principalmente en la mejora de funcionamientos y mecánicas del juego. Para lo cual se ha tratado de analizar de manera profunda todo el desarrollo hasta el momento del proyecto, la búsqueda de errores y posibles mejoras en el mismo y el abordamiento de estos. Dicho trabajo se ha realizado juntamente con otro estudiante del Grado en Ingeniería de Sonido e Imagen de la UPM, Beatriz Ramos García. Ambos hemos trabajado a menudo sobre las mismas áreas del videojuego, abordando principalmente la resolución de problemas de este. Teniendo en cuenta esto, me he centrado principalmente en la lógica del videojuego, trabajando en este sentido de manera compenetrada con Beatriz.

Los objetivos para seguir son la resolución de problemas del videojuego, tanto en lo que a la programación se refiere, entiéndase errores en el código, como al planteamiento y diseño de algunas de las mecánicas o funcionalidades del juego. Una vez se obtenga una versión depurada del proyecto, sin errores, se ha de continuar con el desarrollo del mismo, véase, continuación de la historia y creación de nuevos minijuegos, de manera que se pueda presentar una versión beta plenamente funcional.

Durante las primeras semanas el trabajo consistió en la familiarización con el entorno y el videojuego, poniendo a prueba el mismo mediante el *gameplay*, realizando labores de *testing* para después abordar los problemas encontrados mediante la depuración de código. Dicho trabajo consiste en la localización del origen del error, es decir, qué script y qué líneas de este generan el problema, el planteamiento de una estrategia para resolverlo y la comprobación de la misma hasta obtener un resultado satisfactorio. Posteriormente nuestro trabajo se centró en la creación de un script nuevo que controla dos de los minijuegos del proyecto para conseguir un funcionamiento correcto y fluido, además de adaptarlo finalmente a cada uno de estos minijuegos. Después se trató de abordar un error existente en el videojuego desde hace varios años, el cual consiste en la incorrecta correspondencia entre los movimientos del paciente y los del modelo, traduciéndose

esto en rotaciones erróneas de algunas extremidades de dicho modelo. Para ello se analizó y estudió el trabajo realizado en esta materia por César Luaces, centrándonos en la calibración del modelo y, posteriormente, trabajando juntamente con Eduardo Botija Santamaria, desarrollador de otro videojuego del departamento, introduciendo un nuevo método de calibración implementado por él.

2. OBJETIVOS DE LAS PRÁCTICAS, TAREAS Y ACTIVIDADES REALIZADAS

Inicialmente el objetivo de las prácticas era la continuación de la historia abordando el desarrollo de la segunda parte del *gameplay*. Sin embargo, al iniciar la toma de contacto con el entorno y con el trabajo realizado por los alumnos que han desarrollado anteriormente el videojuego se estableció como prioridad la depuración del proyecto, trabajo ya comenzado por otros estudiantes en el semestre anterior en cuanto a la reestructuración del proyecto y mejora de varias mecánicas.

En este contexto, los objetivos y trabajos realizados en el proyecto se podrían resumir de la siguiente manera: **Familiarización con el entorno** para obtener una comprensión general del proyecto estructurada y poder trabajar de manera ordenada sobre este, **depuración de errores** de código y diseño mediante *testing* y aplicación de estrategias para subsanarlos mediante programación, **implementación de scripts** para la mejora de funcionamientos en algunos minijuegos, **mejora del funcionamiento del juego en lo referente a la Kinect** incidiendo especialmente en las rotaciones indeseadas de las extremidades del modelo.

A continuación, se van a desarrollar los objetivos introducidos anteriormente:

1. **Familiarización con el entorno:**
   "Phiby's Adventure" es un proyecto de tamaño considerable y con objetivos a largo plazo por el que han pasado gran cantidad de alumnos haciendo distintas e importantes aportaciones en diferentes áreas de este. Siendo esta una de las ventajas del videojuego también es una de las desventajas al comenzar a trabajar en él, ya que antes de realizar nuevas ideas o cambios se ha de comprender primero como está estructurado el proyecto y cómo están implementadas cada una de las mecánicas y funcionalidades del juego, lo cual puede resultar considerablemente complejo al principio.

   Así, en esto se centró el trabajo los primeros días en las prácticas, en los cuales se dedicaron esfuerzos principalmente a probar el videojuego a la vez que se estudiaba con mayor fijación los scripts que controlan los funcionamientos que se veían reflejados en el *gameplay* y a la anotación de los errores que se iban encontrando. En este sentido, se comenzó por analizar los primeros scripts que se ejecutan al lanzar la aplicación, los cuales son los relacionados con la carga de los parámetros que necesita cada minijuego desde la web del terapeuta, así como con la creación de los objetos que controlan y almacenan la información relevante entre escenas. Para ello, se utilizó el diagrama de bloques creado por Henar Redondo (anterior participante en el proyecto), en el cual es descrito dicho funcionamiento:

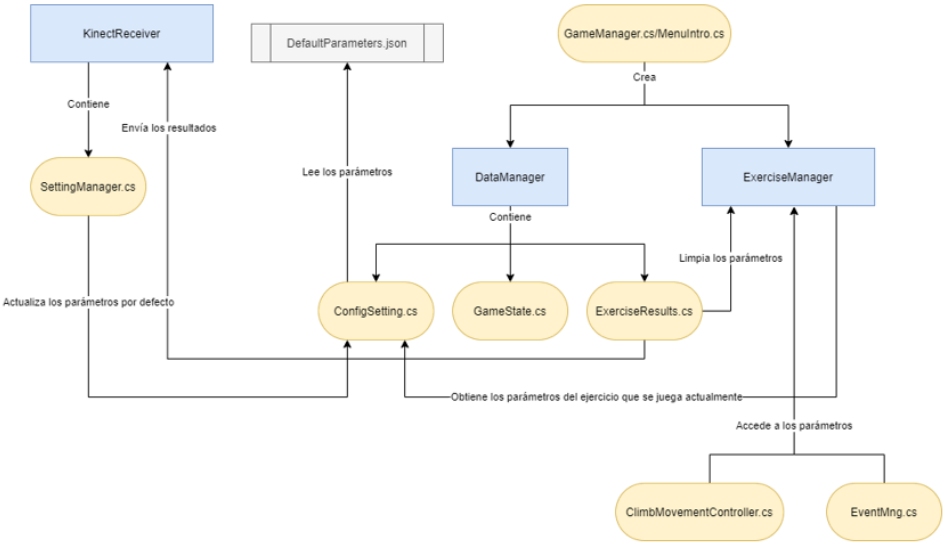

Figura 1.- Diagrama de bloques de obtención de los parámetros necesarios al inicio de la aplicación (Henar Redondo).

Después se abordó el estudio de scripts más específicos que controlan el funcionamiento de las escenas, tanto la principal como las relacionadas con los minijuegos, determinando en qué scripts es controlado el cambio de *checkpoint*, esto es, variable que almacena en qué etapa del juego se encuentra el jugador de las seis existentes hasta el momento. Esta información es de especial utilidad e importancia ya que gracias a ella es posible determinar el valor de numerosas variables y el estado de diversos objetos cuando se cambia de una escena a otra, por lo que su correcto funcionamiento y compresión es crítico.

Como se ha explicado, diversos errores de programación fueron encontrados durante esta fase, por lo que se entró en la siguiente etapa que define el segundo de los objetivos sintetizados en este apartado.

2. **Depuración de errores:**

Como se ha comentado, durante la puesta en contacto con el videojuego se encontraron gran cantidad de errores gracias a la herramienta de depuración ofrecida por Unity “Error Pause” en la ventana *Console*. La estrategia a seguir ha consistido en identificar dichos errores, buscar en qué scripts se generaban y dónde, lo cual no siempre es obvio, y la resolución de los mismos de una manera ordenada. Así, a continuación, se van a explicar algunos de los errores detectados y resueltos más importantes:

- Errores de estructura en el archivo JSON que almacena los parámetros por defecto que son utilizados por los minijuegos en caso de que no hayan sido cargados desde la web.
- Destrucción de objetos de manera desordenada o sin realizar prevención de errores. En algunos casos se han encontrado planteamientos de implementación en los cuales un objeto al cual se accede de manera “periódica” (en un Update por ejemplo) es destruido en otro lugar del script o en otro script del proyecto. Estas acciones de destruir objetos son generalmente realizadas al cambiar de escena, proceso que puede llevar unos segundos en los cuales se pueden suceder gran cantidad de frames y, por tanto, ejecutarse tantas veces la función “Update”

en la cual se quiere acceder al objeto destruido. La estrategia escogida para resolver este tipo de errores ha consistido en crear una función "DisableInvocations" en el script en el que se trata de acceder al objeto que se ha destruido y a la cual se llama en el script en el cual se trata de destruir el objeto antes de realizar esta acción. Esta función pone a true un booleano. Posteriormente en la función "Update" se añade una línea de código que evita que se acceda al objeto en cuestión en caso de que el booleano este activo. Así, mediante este sencillo funcionamiento se consiguen mantener las mecánicas implementadas en un inicio, pero sin errores. Un ejemplo de esto se puede observar en los scripts Rotation.cs y PhibyMixedController.cs. En Rotation.cs se llama a la función "DisableInvocations" del PhibyMixedController.cs previamente a destruir un objeto en la función "finishAndDestroy".

```
public void finishAndDestroy()
{
    //Se destruye el objeto kinectReceiver
    gameObject.GetComponent<UDPReceive>().finalizarConexion();
    udpSend.finalizarConexion();

    /*Next condition lets send a message to the script "PhibyMixedController" and, in this way avoids the access to
      one of kinectReceiver's sons before destroying it (Andrés)*/
    if(GameObject.Find("PhibyKeyboard")!= null)
        GameObject.Find("PhibyKeyboard").SendMessage("DisableInvocations");
    /*Next conditions lets send a message to the script "IKManager" and, in this way avoids the access to
      one of kinectReceiver's sons before destroying it (Andrés)*/
    if(GameObject.Find("AmplificationManagerR")!= null)
        GameObject.Find("AmplificationManagerR").SendMessage("DisableInvocations");
    if(GameObject.Find("AmplificationManagerL")!= null)
        GameObject.Find("AmplificationManagerL").SendMessage("DisableInvocations");


    Destroy(gameObject);
}
```

Figura 2.- Llamada a la función "DisableInvocations" en el script Rotation.cs

```
public bool invocationsOff; //Used by method "DisableInvocation()" to exit from Update() in case of kinect control is enabled when there
                            //is a change of scene in the island.

/*********************************************DISABLEINVOCATIONS*********************************************************************
 *This method avoids an error that occurs when the player tries to access minigame scenes when Kinect control is enabled.
 *This is because when that happens, the Rotation.cs script destroys the kinectReceiver, however you are still trying to access
 *one of kinectReceiver's sons in this script (specifically, you are trying to access the GameObject "HipCenterUp") assigned
 *to the gameObject "positionPhiby".
 *DisabledInvocations() switchs the bool invocationsOff value to true. Thus, if the Kinect control is enabled and invocationsOff is true,
 *Update() performs a return, preventing the positionPhiby gameObject access.
 */
public void DisableInvocations(){
    invocationsOff = true;
}
```

Figura 3.- Función "DisableInvocations" en script PhibyMixedController.cs

```
if(invocationsOff)//To avoid access to gameObject positionPhiby if it's been destroyed.
    return;
```

Figura 4.- Condicional en la función "Update" de PhibyMixedController.cs

- Error al tratar de acceder al script SuperPower.cs asociado en la escena de la isla al objeto "SuperPower". Se observó que la manera en la cual se trataba de acceder a dicho objeto estaba mal planteada. El objeto "SuperPower" se desactiva en la función "Start" del script SuperPower.cs, asociado a sí mismo, en caso de que el checkpoint en el momento de cargarse la isla no sea el cuatro. Sin embargo, el checkpoint cambia del tres al cuatro durante uno de los eventos que suceden en la propia escena de la isla, por lo que el objeto "SuperPower" nunca era reactivado si no se empezaba el juego directamente desde el checkpoint cuatro. Los problemas derivados de esto eran que a ojos del jugador el objeto

"SuperPower", el cual es imprescindible para continuar la historia del juego, no existía, y que cuando algún script como el RockyController.cs o el RockyTrap.cs quería acceder al mismo no podían porque este objeto nunca había sido encontrado.

Para solucionarlo han sido necesarias dos acciones. Por un lado, se ha replanteado la manera en la cual RockyController.cs y el RockyTrap.cs acceden desde su función "Start" al script SuperPower.cs del objeto "SuperPower". Se ha tenido en cuenta que si se inicia el juego desde el principio, como haría el jugador, el objeto "SuperPower" debe estar en primera instancia desactivado. Por tanto, para poder acceder a su script SuperPower.cs se ha hecho hijo este objeto de un objeto contenedor llamado "SuperPowerContainer", el cual no es desactivado nunca. De esta manera se puede seguir accediendo al script SuperPower.cs del objeto "SuperPower" a través de su padre, aunque dicho objeto "SuperPower" esté desactivado. A continuación, se muestran unas capturas del código para que se entienda mejor:

Figura 5.- Parentesco "SuperPower" en ventana Hierarchy.

```
superPowerContainer = GameObject.Find("SuperPowerContainer");
superPower = superPowerContainer.transform.GetChild(0).GetComponent<SuperPower>();
```

Figura 6.- Acceso al script SuperPower.cs del objeto "SuperPower" a través de su padre en la función "Start" del script RockyTrap.cs

Por otro lado, también se ha reactivado el objeto "SuperPower" en la escena para que el jugador pueda hacer uso de él. Esto se ha realizado en el script RockyTrap.cs cuando se cumplen una serie de condiciones relacionadas con el progreso actual de la historia:

```
private void OnTriggerEnter(Collider collider)
{
    if (collider.tag == "Player" && gameManager.checkpoint == 4)
    {
        if (!entered)
        {
            if (!aS.isPlaying && help)
            {
                aS.PlayOneShot(clips[0]);
                gameManager.ShowTip("Phiby, I'm trapped below the rocks and cannot move them!
                superPowerContainer.transform.GetChild(0).gameObject.SetActive(true);
```

Figura 6.- Reactivación del objeto "SuperPower" en el script RockyTrap.cs

- Se encontraron en diversas escenas el uso de scripts que no habían sido diseñados para esa escena en concreto pero que tenían alguna funcionalidad que interesaba aplicar. Esto provocaba que, aunque a priori se conseguía el objetivo de aprovechar una de las funcionalidades concretas del script en cuestión, el script tratase de acceder o encontrar objetos que debían estar en la escena para la que fue diseñado, pero no estaban en la escena en la que se estaba utilizando dicho script. Un ejemplo de esto era que varias escenas de minijuegos tenían un objeto "MiniGameManager" con el script MiniGameManager.cs asociado, el cual fue diseñado para la escena "MiniGamesSimulator", que tiene unos objetos muy concretos en su jerarquía y que algunos de ellos no están presentes en las

escenas de los minijuegos y a los cuales este script trata de acceder. Así, el objetivo de utilizar dicho script en otras escenas era utilizar la función "ExitMiniGame" pero no se tenía en cuenta los problemas que ese planteamiento suponía. Errores como este parecen ser relativamente habituales en el proyecto.

Un ejemplo de cómo se abordó este problema se encuentra en el script MiniGameEnergyController.cs, el cual ahora, en vez de acceder a la función "ExitMiniGame" del MiniGameManager.cs, dispone de una función propia llamada "exitScene" que permite utilizar la misma funcionalidad pero teniendo en cuenta que este script se utiliza tanto en las escenas de minijuegos como en la escena de "MiniGamesSimulator", por lo que diferencia en qué tipo de escena se esta llamando a esta función y actúa en consecuencia llamando a la función que quiere utilizar del script EventMng.cs (objeto que sí se encuentra en las escenas de los minijuegos) en caso de que se trate de una escena de un minijuego o a la función correspondiente del script MiniGameManager.cs en caso de que trate de la escena del simulador. Se adjunta una captura de dicha función:

```
public void exitScene()
{
    if(SceneManager.GetSceneByName("MiniGamesSimulator")==null)
    {
        eventMng = GameObject.Find("EventMng").GetComponent<EventMng>();
        eventMng.SendMessage("exitScene");
    }
    else
    {
        MiniGameManager = GameObject.Find("MiniGameManager").GetComponent<MiniGameManager>();
        StartCoroutine(MiniGameManager.ExitMiniGame());
    }
```

Figura 7.- Función "exitScene" del script MiniGameEnergyController.cs

- Diversos errores en lo relativo al acceso a variables u objetos en bucles sin tener en cuenta el borrado o la variación de los mismos, errores en lo referente a la sintaxis del lenguaje de programación utilizado, etc.

3. Implementación de scripts:

Uno de los ejercicios de los que dispone el videojuego consiste en conseguir que el personaje escale una cierta altura mientras el paciente hace dicho movimiento en tiempo real. Este ejercicio cuenta con dos variantes, la primera de ellas en la escena "ClimbCage", la cual es muy permisiva en lo que a movimientos del paciente se refiere, ya que no exige un orden en el movimiento de los brazos ni bajar uno de ellos antes de subir el otro, por ejemplo. La segunda versión o variante de este ejercicio se utiliza en la escena "ClimbAppleTree", la cual es más exigente ya que los movimientos del paciente deben ser ordenados, deben abarcar un rango de movimiento mínimo y no admite como válido un movimiento si se suben ambos brazos a la vez entre otras cosas.

Ya existían scripts que trataban esta funcionalidad, sin embargo, los resultados no eran muy buenos en lo que a fluidez y jugabilidad se refiere, especialmente en el script utilizado en la escena "ClimbAppleTree".

Así, se han creado dos scripts llamados ClimbingTrees.cs (utilizado en "ClimbAppleTree") y ClimbCage_v2.cs (utilizado en "ClimbCage"). Ambos disponen de una funcionalidad adaptada a cada escena según los requisitos que se han explicado anteriormente, pero el

mecanismo que controla el movimiento es el mismo. Se han utilizado dos objetos vacíos con sus correspondientes *colliders* esféricos que constituyen el objetivo o punto al cual debe llegar el modelo con la mano para poder escalar, como si de las barras de una escalera se tratara. Cada uno de estos objetos tiene asociado un script ClimbingTrees.cs o ClimbCage_v2.cs según la escena, los cuales se comunican entre si permitiendo la coordinación y control de los movimientos de ambas manos, lo cual es de especial importancia en el caso de ClimbingTrees.cs.

Como es de esperar, la base de este funcionamiento se encuentra en el uso de las funciones "OnTriggerEnter" y "OnTriggerExit". De esta manera en ClimbingTrees.cs se emplean varios booleanos para comprobar qué mano se ha levantado previamente, si la otra mano está levantada o si el modelo está ascendiendo, además de gestionar el recuento de manzanas, el gasto de la energía y la aparición de objetos manzana a la altura a alcanzar como realimentación para el jugador.

4. **Mejora del funcionamiento del juego en lo referente a la Kinect:**

   El último de los objetivos a los que se han hecho frente y el que más tiempo ha llevado ha sido el de solucionar los problemas relacionados con rotaciones indeseadas en las extremidades del modelo, los cuales están relacionados con cómo se traducen los movimientos del jugador capturados con la Kinect a los movimientos que realiza el modelo en el juego. Es un problema muy complejo y que se lleva arrastrando desde el inicio del proyecto.

   La primera estrategia para tratar de abordar este problema y entenderlo en mayor profundidad fue el estudio del proceso de calibración definido por César Luaces en el Anexo 2 de su Trabajo de Fin de Grado. Para realizar una primera aproximación a este proceso de calibración se creó un proyecto aparte con un modelo de prueba que contiene todas las articulaciones posibles. Dicho proceso consiste en obtener unos valores máximos y mínimos en lo que a la posición y rotación de la extremidad sobre la que se esté trabajando se refiere. Para ello, se dispone de un objeto "target" hacia el cual se orienta la extremidad a la que esté asociado. Este funcionamiento está implementado en el script IKManager.cs creado por César Luaces, el cual aplica la estrategia de cinemática inversa. Así, moviendo el "target" es posible hacer que la extremidad en cuestión lo siga hasta colocarse en las posiciones deseadas, que en este caso serán las posiciones extremas que pueda tomar la extremidad tanto horizontal como verticalmente, por ejemplo, si se trata de un brazo este se deberá colocar extendido hacia delante y hacia atrás para hacer la calibración horizontal, y extendido hacia arriba y hacia abajo en el caso de la calibración vertical. Los valores que se obtienen en cada posición consisten en una posición y una rotación con respecto a un objeto definido como referencia. De esta manera, para la calibración horizontal por ejemplo se obtienen cuatro valores, dos de posición y dos de rotación, de donde se determinan los máximos y los mínimos tanto de posición como de rotación. Una vez hecho esto, dichos valores se asocian a las variables correspondientes del script LauncherAmp.cs y determinarán la calibración de la extremidad sobre la que se esté trabajando.

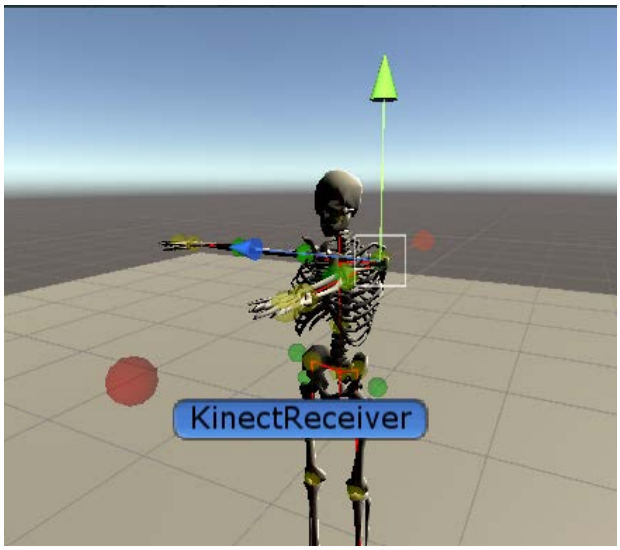

Figura 8.- Modelo de prueba en posición extremo horizontal hacia delante durante el proceso de calibración.

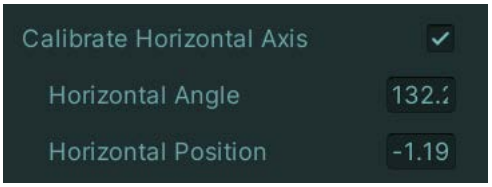

Figura 9.- Valores de calibración obtenidos en la posición de la Figura 8.

Este procedimiento fue llevado a cabo tanto en el proyecto de prueba mencionado anteriormente como en el proyecto del videojuego sobre el modelo de Phiby. En ambos casos los resultados obtenidos no fueron satisfactorios ya que no se consiguió obtener una correspondencia entre el movimiento del jugador y del modelo. Sin embargo, sí se acotó el problema, el cual se encuentra en el script IKManager.cs, que, debido a su planteamiento, genera algunas rotaciones indeseadas, en el caso del modelo del videojuego de este proyecto, en el brazo izquierdo.

Figura 10.- Ejemplo de valores máximos y mínimos obtenidos para el modelo de Phiby e introducidos en la ventana Inspector del script LauncherAmp.cs.

Este proceso de calibración y método de movimiento de las extremidades del modelo es el que se emplea actualmente en las escenas “ClimbCage” y “ClimbAppleTrees” del proyecto.

La segunda estrategia para tratar de abordar el problema descrito ha consistido en aplicar el método de calibración y de movimiento de las extremidades del modelo implementado por Eduardo Botija Santamaría. Dicho método se basa en los scripts

CalibrationController.cs y testIK.cs, el cual también está basado en la técnica de cinemática inversa, pero ofreciendo un funcionamiento más sencillo y preciso. Sin embargo, este método ha sido diseñado para un modelo humanoide, por lo que se han generado varios problemas al aplicarlo a Phiby, ya que este modelo no es humanoide. Como resultado se han generado unas rotaciones indeseadas en las articulaciones correspondientes a los brazos del modelo.

En cuanto al método de calibración implementado por Eduardo, este consiste en realizar unas mediciones previas antes de jugar en las que el jugador ha de realizar unos movimientos específicos que, al ser captados por la Kinect, servirán al juego como referencia. Así, antes de empezar a jugar se ha de pulsar la tecla "E" y el jugador deberá mover el tronco hacia delante, detrás y hacia ambos lados. Posteriormente se ha de pulsar la tecla "R" y el jugador deberá realizar rotaciones con ambos brazos. Finalmente se ha de pulsar la tecla "T" para empezar a jugar.

A continuación, se adjunta una captura de la ventana "Inspector" del script que implementa la calibración mencionada:

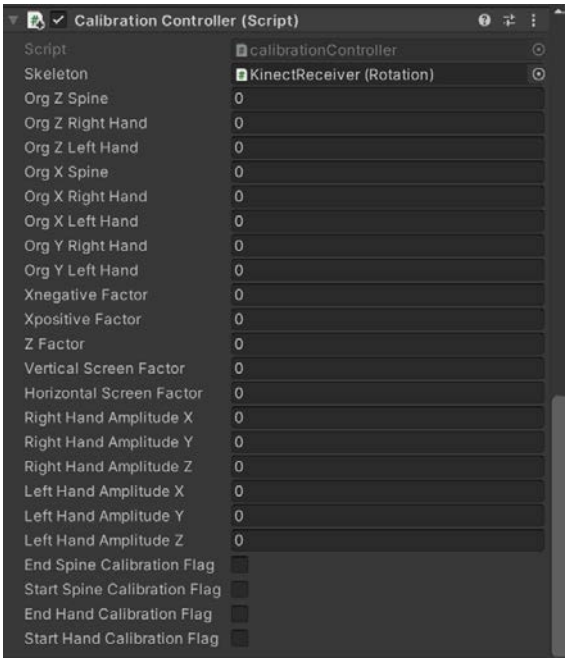

Figura 11.- Parámetros que se obtienen en tiempo real durante la calibración implementada en CalibrationController.cs

Esta segunda estrategia ha sido aplicada en la escena "ChopWoodPhiby".

En dicha escena también ha sido necesario realizar varios cambios en la misma en lo referente a las posiciones de los troncos y del personaje. Esto es debido a que para obtener un correcto funcionamiento de la cinemática inversa el modelo debe estar orientado hacia el centro de coordenadas.

En versiones previas, en dicha escena el modelo estaba rotado en el eje Y, lo cual era fuente de diversos problemas debido a esta razón. Así, para orientar correctamente al personaje ha sido necesario cambiar de posición también los troncos, lo que ha conllevado la actualización de las posiciones a las cuales se desplazan los pedazos de los troncos cuando son cortados según las animaciones creadas por Marta Sanz.

Como se ha dicho, en dicha escena se aplica el método de calibración de Eduardo, el cual precisa de un tiempo previo para realizarla, por lo que ha sido necesario eliminar de manera provisional la cuenta atrás presente en el script ChopMovementController.cs. En este sentido, se está trabajando para la llevar a cabo, mediante programación, el almacenamiento de los datos de calibración en un JSON al cual se podrá acceder al comenzar el minijuego, de manera que no sea necesario realizar la calibración siempre que se acceda al mismo.

Por último, cabe destacar que también se ha implementado un método para generar la salida del minijuego una vez se han obtenido el número máximo de troncos, ya que está funcionalidad no estaba implementada. Dicho método es provisional y consiste en una sencilla función llamada “ChopSum” que suma 1 a la variable “numberOfChops” del script ChopMovementController.cs. Esta función se encuentra implementada en ChopMovementController.cs y es llamada desde LogAnimated.cs cada vez que se lanza la animación se cortar un tronco. Su sentido radica en que cuando la variable “numberOfChops” alcanza el valor máximo de troncos cortados se genera la salida del minijuego.

3. EVOLUCIÓN CRONOLÓGICA DE LAS ACTIVIDADES REALIZADAS Y LOGROS OBTENIDOS DURANTE EL PERÍODO,

A continuación de presenta la cronología de las actividades llevadas a cabo durante las prácticas realizadas, las cuales han sido descritas en el apartado anterior:

Nombre actividad	Fecha inicio	Duración	Fecha fin
Familiarización con el entorno	30-sep	20	20-oct
Depuración de errores	11-oct	15	26-oct
Implementación de scripts	26-oct	2	28-oct
Mejora del funcionamiento (Kinect)	28-oct	53	20-dic

Figura 12.- Tabla cronológica de las actividades desarrolladas durante las prácticas.

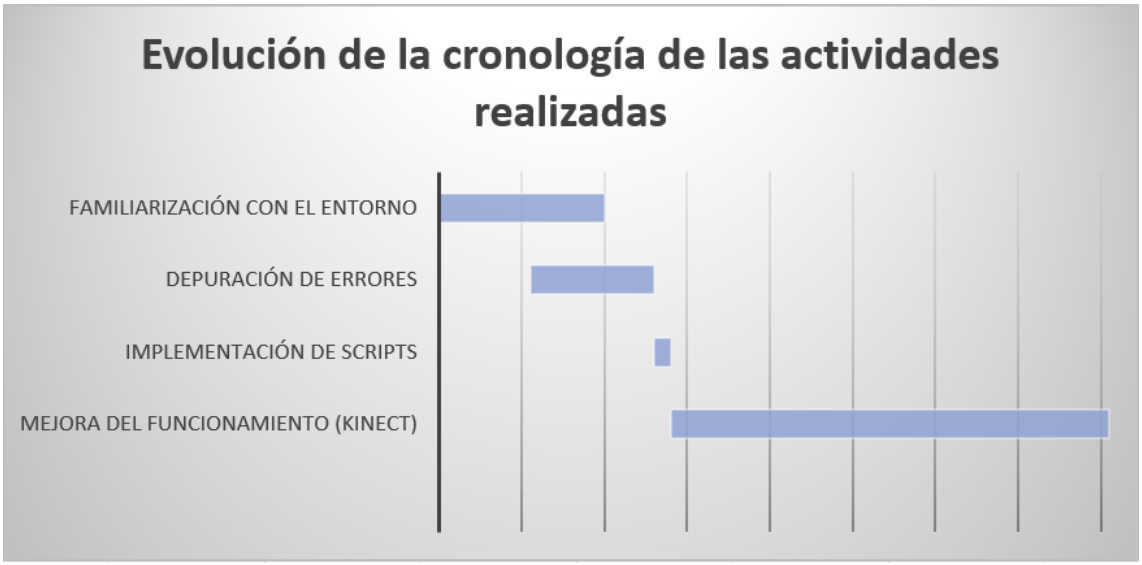

Figura 13.- Diagrama de Gantt de las actividades realizadas.

4. TECNOLOGÍAS Y MEDIOS TÉCNICOS UTILIZADOS

Entre las tecnologías y los medios utilizados para el desarrollo de las prácticas destacan los siguientes:

- **Unity:** Unity es un motor de desarrollo de videojuegos, el cual dispone de un software que tiene una serie de rutinas de programación que permiten el diseño, la creación y el funcionamiento de un entorno interactivo. Entre sus funcionalidades destacan un motor gráfico para renderizar gráficos tanto en 2D como en 3D, un motor físico que simula las leyes de la física, animaciones, sonidos, inteligencia artificial, scripting, etc.
  Así, Unity es un software que centraliza todo lo necesario para poder desarrollar videojuegos, permitiendo crear estos mediante un editor visual y programación vía scripting, utilizando como lenguaje C# en el caso de este proyecto y Visual Studio como aplicación para implementar los scripts.
  Cabe destacar que una de las grandes ventajas de Unity es la amplitud de la información a servicio del usuario de la que se dispone a través de la Unity Scripting API, la cual ha sido utilizada a menudo durante el transcurso de las prácticas.
- **Kinect Xbox One**: Kinect es una línea de dispositivos de reconocimiento de movimiento producido por Microsoft. En concreto, en este proyecto se ha utilizado la Kinect para Xbox One, la cual utiliza una cámara gran angular con tecnología "tiempo de vuelo", la cual es una técnica para estimar distancias de cuerpos calculando el tiempo transcurrido entre la emisión y recepción de un haz de luz infrarrojo. Procesa 2 Gbits de información por segundo para capturar su entorno, cuenta con una resolución de 1080p, captura 30 imágenes por segundo, y puede detectar la posición y orientación de 25 articulaciones, así como la velocidad de los gestos y movimientos del jugador.
- **K2UM (Kinect to Unity Middleware)**: K2UM es un programa que sirve de intermediario entre la cámara de captura de movimiento, es decir, la Kinect, la plataforma web terapéutica Blexer-med y el videojuego en cuestión, permitiendo así el control de este mediante movimientos corporales a partir de la transformación de la información capturada por la cámara en paquetes UDP con la información del movimiento y rotación de cada articulación. Fue desarrollado por César Luaces.

5. COMPETENCIAS Y HABILIDADES ADQUIRIDAS CON LAS PRÁCTICAS

En cuanto a las competencias y habilidades adquiridas durante el periodo de prácticas se destacan las siguientes:

- C_GEN_02: Capacidad de búsqueda y selección de información, de razonamiento crítico y de elaboración y defensa de argumentos dentro del área.
- C_GEN_04: Capacidad de abstracción, de análisis y de síntesis y de resolución de problemas.
- C_GEN_05: Capacidad de trabajo en equipo y en entornos multidisciplinares.
- C_GEN_08: Capacidad de organización, planificación y de toma de decisiones.
- C_GEN_11: Habilidades para la utilización de las Tecnologías de la Información y las Comunicaciones.
- C_BAS_02: Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería.
- C_TEL_01: Capacidad para aprender de manera autónoma nuevos conocimientos y técnicas adecuados para la concepción, el desarrollo o la explotación de sistemas y servicios de telecomunicación.
- C_TEL_08: Capacidad de realizar programación en tiempo real, concurrente, distribuida y basada en eventos, así como el diseño de interfaces persona-computador.
- C_SI_05: Capacidad para crear, codificar, gestionar, difundir y distribuir contenidos multimedia, atendiendo a criterios de usabilidad y accesibilidad de los servicios audiovisuales, de difusión e interactivos.

6. CONCLUSIONES

El periodo de prácticas externas en el centro de investigación ha sido enriquecedor en varios sentidos. Primero por el hecho de trabajar en un proyecto relativamente grande en el que han trabajado numerosos alumnos, lo cual me ha permitido cambiar mi percepción de lo que un proyecto de investigación supone, dándome un enfoque más realista y mejorando mis capacidades analíticas y de trabajo en equipo.
En concreto, agradezco poder haber participado en el desarrollo de un videojuego en Unity, ya que es un entorno que realmente me gusta y, aunque ha habido momentos de frustración frente a la cantidad de código y a la complejidad de este, puedo decir que he disfrutado la experiencia. Además, el hecho de que se trate de un proyecto que puede repercutir positivamente de manera directa en la salud y vida de los usuarios, siempre me ha parecido uno de los grandes atractivos del mismo.
Si bien es cierto que mi idea inicial no era la de asumir la depuración de errores de manera continuada durante los meses que han durado las prácticas, soy plenamente consciente de que era imprescindible para poder continuar el desarrollo del videojuego correctamente y por ello estoy contento de poder haber aportado mi ayuda en este sentido, aunque no se hayan podido resolver todos los problemas afrontados, especialmente los relacionados con la Kinect, que han sido los qué mas tiempo no han llevado y no ha sido posible conseguir los objetivos que nos propusimos respecto a ellos.
Por último, también quisiera destacar el buen ambiente de trabajo que existe en el laboratorio, el cual se debe en gran medida a la confianza y comprensión que Martina Eckert deposita en nosotros. Estoy muy agradecido por ello, tanto a Martina como al resto de compañeros, especialmente a Beatriz y Eduardo.