universidad de león

Escuela de Ingenierías I. I.

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

CONEXIÓN KINECT - UNITY3D
a través de DLL para la creación de una interfaz gestual y mediante comandos de voz

Autor: Jorman Hernández Armas

Tutor: Fernando Jorge Fraile Fernández

RESUMEN

La idea básica consiste en conectar el motor de juego Unity3D con el dispositivo kinect de Microsoft para el reconocimiento tanto de movimientos corporales como de voz humana.

Unity3d[1] (o **Unity**) es un Motor 3D para el Desarrollo de Videojuegos creado por Unity Tecnologies[2]. Está disponible para la Plataforma **Windows** y **Mac OS X**, y permite crear juegos para Windows, Mac, Xbox 360, PlayStation 3, Wii, iPad, iPhone y también para la plataforma Android. Gracias al Plug-In Web de Unity, también se pueden desarrollar juegos de Navegador, para Windows y Mac, pero no Linux. Además de su uso para crear juegos, Unity permite una vista guiada que permite usarse también para diseños arquitectónicos y animaciones 3D.

Kinect[3] **para Xbox 360**, o simplemente **Kinect** es "un controlador de juego libre y entretenimiento" creado por Alex Kipman, desarrollado por Microsoft para la videoconsola Xbox 360, y desde junio de 2011 para PC a través de Windows 7 y Windows 8. Kinect permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que **reconoce gestos**, **comandos de voz**, y objetos e imágenes.

En definitiva, el objetivo principal es hacer un **plugin**, wrapper o envoltorio y así poder **usar** el motor de juego de **Unity3d** con los gestos y órdenes de voz a través del dispositivo **Kinect**.

[1] http://en.wikipedia.org/wiki/Unity_(game_engine)
[2] http://unity3d.com/
[3] http://es.wikipedia.org/wiki/Kinect

ÍNDICE

Contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

No disponemos de tablas en el documento

1. Introducción

1.1 ¿Qué son los Videojuegos?

Según el Diccionario de la Real Academia Española un **videojuego**[4] es un "dispositivo electrónico que permite, mediante mandos apropiados, simular juegos en las pantallas de un televisor o de un ordenador".

Los distintos medios que tenemos para jugar videojuegos van desde los celulares, pasando por los tablets y llegando hasta las poderosas consolas de mesa y PCs.

1.1.1 Categorías

1. **Core Games,** corresponde a aquellos juegos cuyos valores de producción son los mayores de la industria. Los tiempos de desarrollo varían según el tipo de juego, pero la mayoría de este tipo de juegos se caracterizan por poseer música, efectos visuales y una narrativa que van acorde con las **últimas herramientas tecnológicas del momento**. La característica más importante de este tipo de juegos, es un profundo e intenso *gameplay* (forma en que el usuario interactúa con el jugador), haciendo que el jugador necesite de concentración y un continuo seguimiento de lo que está ocurriendo en el juego. Algunos títulos que se pueden nombrar: *Call of Duty, Gears of War, Unreal Tournament, Half-Life, Counter-Strike*, entre muchos otros.

Figura 1.1.- Half-life Vs Halo

2. **Casual Games,** dirigido principalmente a jugadores casuales. Se caracterizan por poseer un ***gameplay* sencillo de entender**, de modo que

[4] http://tecnoyucas.blogspot.com.es/2011/05/desarrollo-de-videojuegos-aspectos.html

capte la atención de usuarios que no frecuentan videojuegos. Algunos ejemplos serían la mayoría de los juegos de explorador, como los que hay en Facebook. El Nintendo Wii y la mayoría de los juegos de esta consola, se caracterizan por la creación de este tipo de juegos.

Figura 1.2.- Juagando a la Wii

3. **Serious Games,** son juegos con alguna **finalidad** que va más allá de sólo entretener, pudiendo ser la de **educar**, tocar temas de **medicina** o hasta entrenamiento militar. Ejemplo de este tipo de juegos podría ser la serie de *Flight Simulator* donde la idea es emular las acciones de un piloto de avión, o el juego online que simula compras y subastas de la bolsa *Virtual Stock Exchange*.

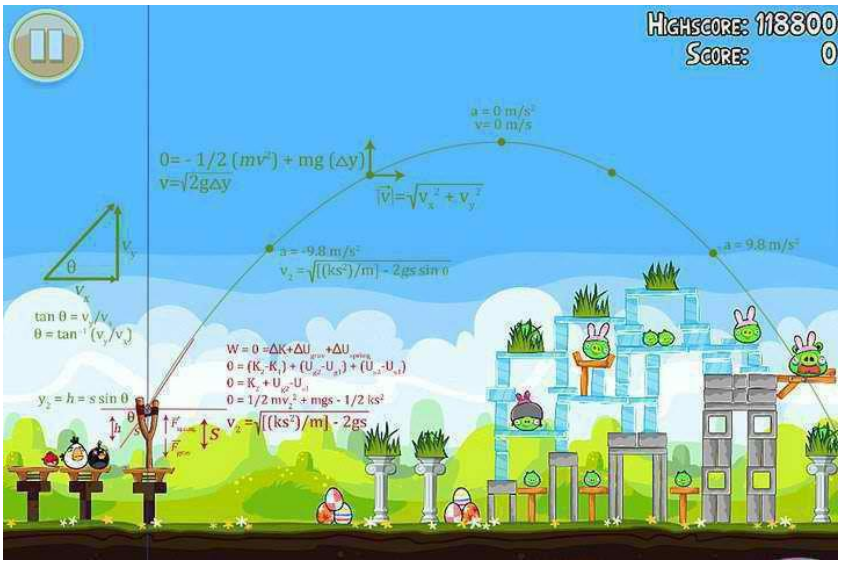


Figura 1.3.- Juego Angry Birds orientado a la física

1.1.2 Formas y engines

De las **consolas** de videojuegos podemos hablar del eje principal de consolas: el Xbox 360 de Microsoft, el Play Station 3 de Sony, el Wii de Nintendo y las PC. Del mercado de videojuegos **móviles** y que entran en la clasificación 'Core' tenemos el Nintendo DS, el Play Station Portable (PSP), y últimamente el

mercado de las tablets, entre las cuales se nombra el poderoso hardware del Ipad o tablets en general.

Según Wikipedia, **los "engines" o "motores" para la creación de videojuegos** "...es un término que hace referencia a una serie de rutinas de programación que **permiten el diseño, la creación y la representación** de un videojuego".

Los engines son representados primeramente por aplicaciones o editores visuales que nos permiten interactuar con dicho software y así crear videojuegos. Para el desarrollo de videojuegos es más sencillo trabajar desde un engine, debido a que tiene ciertos procedimientos pre-elaborados y que, de otra forma, sólo nos quitaría más tiempo llegar a desarrollarlos.

Figura 1.4.- Ejemplos de algunos motores en el mercado

1.1.3 Justificación

Los **videojuegos son** considerados **arte y ciencia**. La industria de los videojuegos tiene más recaudación que la industria del cine y la música juntas. El desarrollo de videojuegos es un proceso complejo, a veces extenso, pero muy creativo y divertido. Para llegar a tener éxito en el medio, es importante desarrollar conocimientos en muchas áreas de desarrollo y programación, así como también en lo referente a diseño y modelación de objetos en tercera dimensión (3D), física avanzada para movimientos y desplazamientos en el

espacio, y hasta lograr manejar el desarrollo de una historia tal cual se hace en las películas y en los libros, o lo que es igual, hacer un guión… Pero antes que nada se necesita **creatividad**.

1.2 ¿Qué es Unity3D?

Unity 3D[5] es una herramienta que nos ayuda a **desarrollar videojuegos para diversas plataformas** mediante un editor y scripting para crear videojuegos con un acabado profesional. Esta herramienta está accesible al público en diferentes versiones, gratuita y profesional, cada cual con sus ventajas y limitaciones. No sólo permite hacer juegos, también es posible usarlo para hacer contenidos interactivos en 3D de una manera **"rápida" y "sencilla"**.

Figura 1.5.- Imagen sacada de la página principal de unity3d.com

La versión gratuita está enfocada para desarrollar videojuegos para **PC, Mac y Web**, este último a través de un plugin para su visionado. Sin embargo, además de estas versiones "básicas", existen añadidos que permiten trasladar nuestro desarrollo a dispositivos móviles. Con Unity además de para web, se puede desarrollar para Xbox360, Wii, IOS y Android. Todas estas plataformas necesitan un desembolso para poder publicar en su "market" y esta exportación a la misma sólo se puede hacer con el Unity Profesional (que es de pago).

[5] http://www.genbetadev.com/herramientas/unity-3d-desarrollo-de-videojuegos-para-ios-y-android-gratis-hasta-el-8-de-abril

Figura 1.6.- Logo de Unity3D

Unity3D está provisto de un **editor visual** muy útil y completo donde, mediante unos pocos clicks, podremos importar nuestros modelos 3D, texturas, sonidos, etc. para después ir trabajando con ellos. Además, incluye la **herramienta de desarrollo MonoDevelop** (e incluso se puede usar Visual Studio de Microsoft) con la que podremos crear scripts en JavaScript, C# y un dialecto de Python llamado Boo con los que extender la funcionalidad del editor, utilizando las API que provee y la cual encontramos documentada junto a tutoriales y recursos en su web oficial.

1.2.1 La interfaz de Usuario de Unity3D

Existen principalmente 5 áreas de la interfaz[6] de Unity3D.

Figura 1.7.- Interfaz de Unity3D

[6] http://es.scribd.com/doc/49807377/Empezando-en-Unity3D-By-index

Vista de Escena (1)

La escena es el área de construcción de Unity donde construimos visualmente cada escena de nuestro juego.

Vista de Juego (2)

En la vista de juego obtendremos una previsualización de nuestro juego. En cualquier momento podemos reproducir nuestro juego y jugarlo en esta vista

Vista de Jerarquía (3)

La vista de jerarquía contiene todos los objetos en la escena actual.

Vista de Proyecto (4)

Existen librerías en Unity que llamamos assets. Estás librerías o assets tienen scripts o comportamientos ya hechos como las explosiones, la cámara de tercera persona, etc. Se pueden importar objetos 3D de distintas aplicaciones a la librería, así como importar texturas y crear otros objetos como Scripts o Prefabs que se almacenarán aquí. Todos los assets que se importen en nuestro juego se almacenarán aquí para que puedan ser usados en nuestro juego.

Vista de inspector (5)

La vista de inspector sirve para varias cosas. Si se selecciona se mostrarán las propiedades de dicho objeto donde se puede personalizar varias características del mismo. También contiene la configuración para ciertas herramientas como la herramienta de terrenos.

1.2.2 Características de Unity3D[7]

1. IDE fácil de manejar, en donde están los principales elementos de la aplicación.
2. Funciona en múltiples plataformas.
3. Assets compatibles con otras aplicaciones de modelado 3D, cómo la serie de Autodesk, Maya y 3ds Max.
4. Soporta shaders que pueden ser programados o importados, así como el uso de texturas.
5. Soporta el engine de movimientos físicos de Nvidia, PhysX.
6. Creación de juegos multiplayer a través de LAN.
7. Programación de scripts a través de UnityScript (muy similar a JavaScript), C# y Boo.

---

7 http://unity3d.com/unity/licenses.html?mkt_tok=3RkMMJWWfF9wsRonvaTOZKXonjHpfsX57ukqX6a0hIkz2EFye%2BLIHETpodcMTctqMa%2BNFAAgAZVnyRQFD%2B6caJkT%2Fg%3D%3D

1.3 ¿Qué es Kinect?

Kinect[8] es un sensor que permite **reconocer los movimientos del cuerpo** y del rostro del usuario, el cual originalmente salió en el año 2010 como accesorio adicional para las consolas de videojuegos Xbox360 de Microsoft. De esta forma, Kinect **elimina el uso de controles** para que el nuevo control sea el propio cuerpo del jugador. Sin lugar a dudas, este pequeño dispositivo generó una revolución en el concepto del juego en estos tiempos, ya que sin la necesidad de controles, siendo el propio cuerpo el encargado de enviar los movimientos a lo largo del juego, el jugador se envuelve más en el rol brindando experiencias de entretenimiento extraordinarias. Mientras otros, se ocupan en poner más accesorios en las manos de los jugadores, Kinect hace que la última barrera entre el jugador y las experiencias desaparezca.

Figura 1.8.- Dispositivo Microsoft Kinect

El nombre de Kinect viene inspirado por la palabra "Kinetic", en español "Cinético", que está relacionado al hecho de estar en movimiento (del inglés "Motion") y conectado (del inglés "Connect"). Por lo tanto, se asocia a *cómo involucra nuestro movimiento y de qué manera nos conecta*.

[8] http://www.wix.com/animusproject/web/apps/blog/una-interfaz-diferente-1

Figura 1.9.- Imagen obtenido de la página de xbox http://www.xbox.com/es-ES/kinect

1.3.1 ¿Qué posee Kinect?

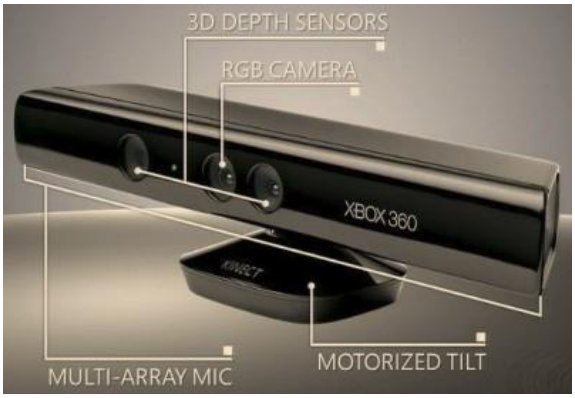


Figura 1.10.- Componentes de la Kinect

Kinect está compuesto de **sensores de profundidad**, una **cámara RGB**, un conjunto de **cuatro micrófonos**, un motor de inclinación y un chip de control.

- Sensores de profundidad

Los sensores de profundidad están compuestos de dos partes, una cámara infrarroja de 640x480 pixels de resolución a 30FPS ((IR CMOS (Complementary Metal Oxide Semiconductor)) Microsoft / X853750001 / VCA379C7130) y un proyector infrarrojo ((IR Projector) OG12 / 0956 / D306 / JG05A).

- Cámara RGB

La cámara RGB es una cámara VGA de 640x480 pixels de resolución a 30FPS ((Color CMOS) VNA38209015).

- Conjunto de micrófonos

Posee un conjunto de cuatro micrófonos, tres posicionados del lado derecho y el cuarto posicionado del lado izquierdo.

- Motor

Posee un motor de inclinación en la base del dispositivo. Los ángulos de inclinación son entre -27 y 27 grados.

- Chip PS1080

El "cerebro" de Kinect se encuentra alojado bajo un chip creado por PrimeSense[9], Dicho chip es el PS1080 SoC (System on a Chip) y el mismo incluye un puerto USB 2.0.

1.3.2 ¿Cómo funciona la Kinect?

El motor de inclinación provisto por la Kinect, en su base, le permite la movilidad del sensor para posibilitar el seguimiento del jugador. Esto se debe a que no todos los lugares de juego son iguales ni tampoco lo son el ancho de los televisores. Este motor permite, por lo tanto, el movimiento del sensor hasta en 27 grados.

Por otro lado, los micrófonos provistos por Kinect permiten captar conversaciones y comandos de voz, distinguiendo la voz del usuario de los sonidos ambiente. Dentro de la tecnología, detrás de la captura de audio, es importante resaltar que los micrófonos de Kinect son abiertos y hacen las capturas en todas las direcciones. De esta forma, se logra capturar sonidos a distancias considerables, que siendo de otra manera solo "escucharía" al televisor.

[9] http://www.primesense.com/

Figura 1.11.- Reconocimiento de voz

El "cerebro" de Kinect el chip **PS1080**, es quien se encarga de todos los controles del sistema. Esto es, del proyector de infrarrojos, del procesamiento de la información que cada cámara recoge y de las entradas de audio. Recabando esta información, este chip es el encargado de generar los mapas de profundidad y de color que luego serán transferidos mediante el puerto USB 2.0 al host destino que requiera dicha información para su procesamiento.

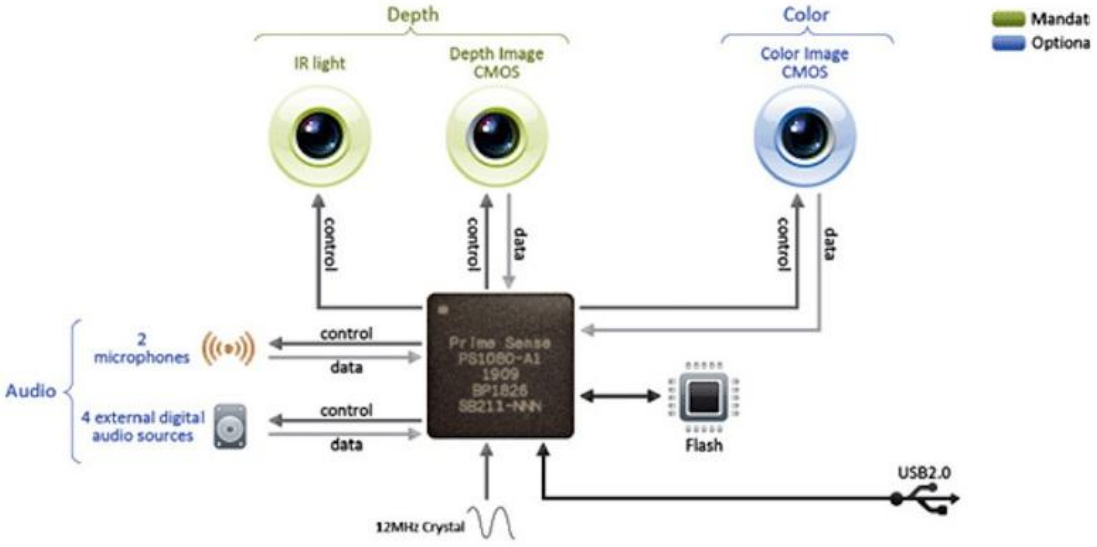


Figura 1.12.- Chip PS1080

La forma de operar de Kinect para generar el mapa de profundidad es sencilla: primero el proyector infrarrojo transmite un patrón de rayos infrarrojos invisibles, el cual convierte todo en pequeños puntos infrarrojos. Luego este patrón es percibido por la cámara infrarroja tras rebotar contra los distintos objetos presentes en la escena. A causa de este rebote el patrón sufre modificaciones.

Dado que el patrón de rayos infrarrojos es conocido, y que además la cámara y el proyector tienen una calibración también conocida, se puede asociar cada punto emitido con el recibido.

A partir de las distorciones (en tamaño y ángulo) percibidas es que el PS1080 quien calcula la distancia a los objetos y genera la imagen de profundidad.

Esta técnica utilizada se le denomina "luz estructurada" y se basa en estudiar la deformación que sufre un patrón de luz al ser intersectado por cualquier objeto. Luego mediante la cámara RGB se genera el mapa de color.

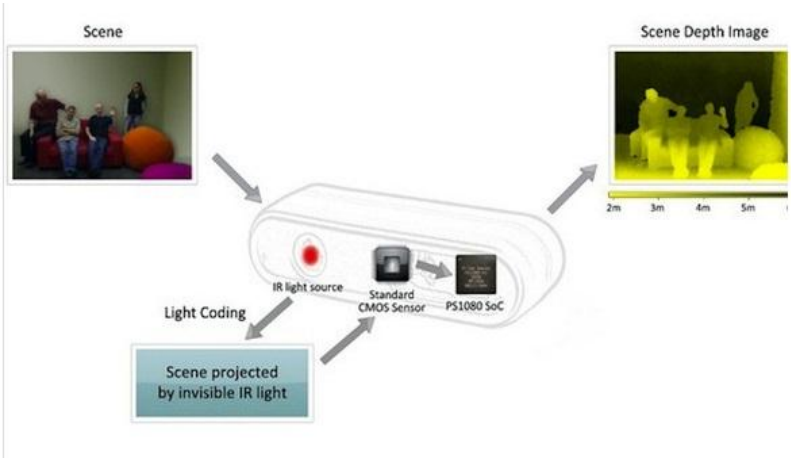


Figura 1.13.- Cámara RGB e Imagen de profundidad

De esta forma, todo el entorno es almacenado en dos mapas, uno de color y otro de profundidad. Además, Kinect cuenta con algoritmos para sacar el mejor provecho de la información que recibe y así renderizar en tiempo real las imágenes generadas en tres dimensiones.

Usando el proyector de rayos infrarrojo también se soluciona parcialmente el problema de la interferencia por luz ambiental, ya que, el sensor no está diseñado para registrar la luz visible y por lo tanto no tendrá tantos falsos positivos o lecturas erróneas.

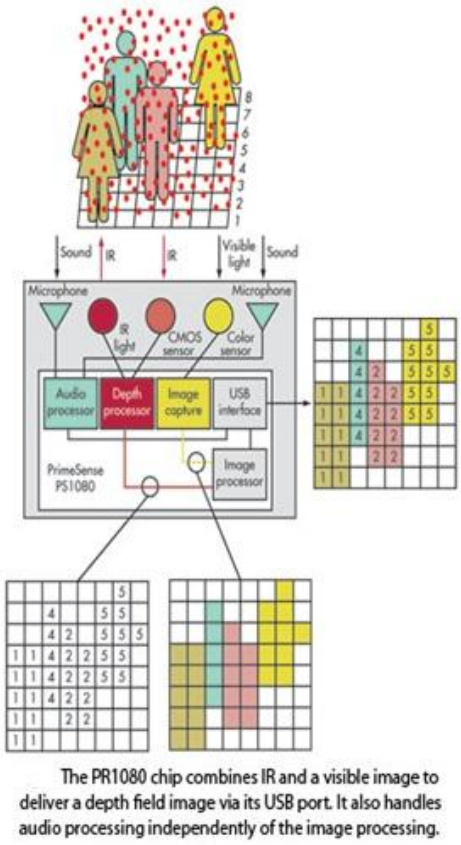


Figura 1.14.- Funcionamiento de Kinect

De esta forma, el software residente en el "host" puede reconocer mediante estos mapas cosas como qué es lo que se está moviendo, reconociendo de entre esos objetos a los seres humanos, para luego, llevar a cabo, el reconocimiento de las partes del cuerpo humano, como brazos, piernas, articulaciones etc. Por lo tanto, el cuerpo será reconocido sin importar la intensidad de la luz ambiente, y no solo se detectará el movimiento de las manos y muñecas, sino de todo el cuerpo, es decir, brazos, piernas, rodillas, cintura, cadera, etc.

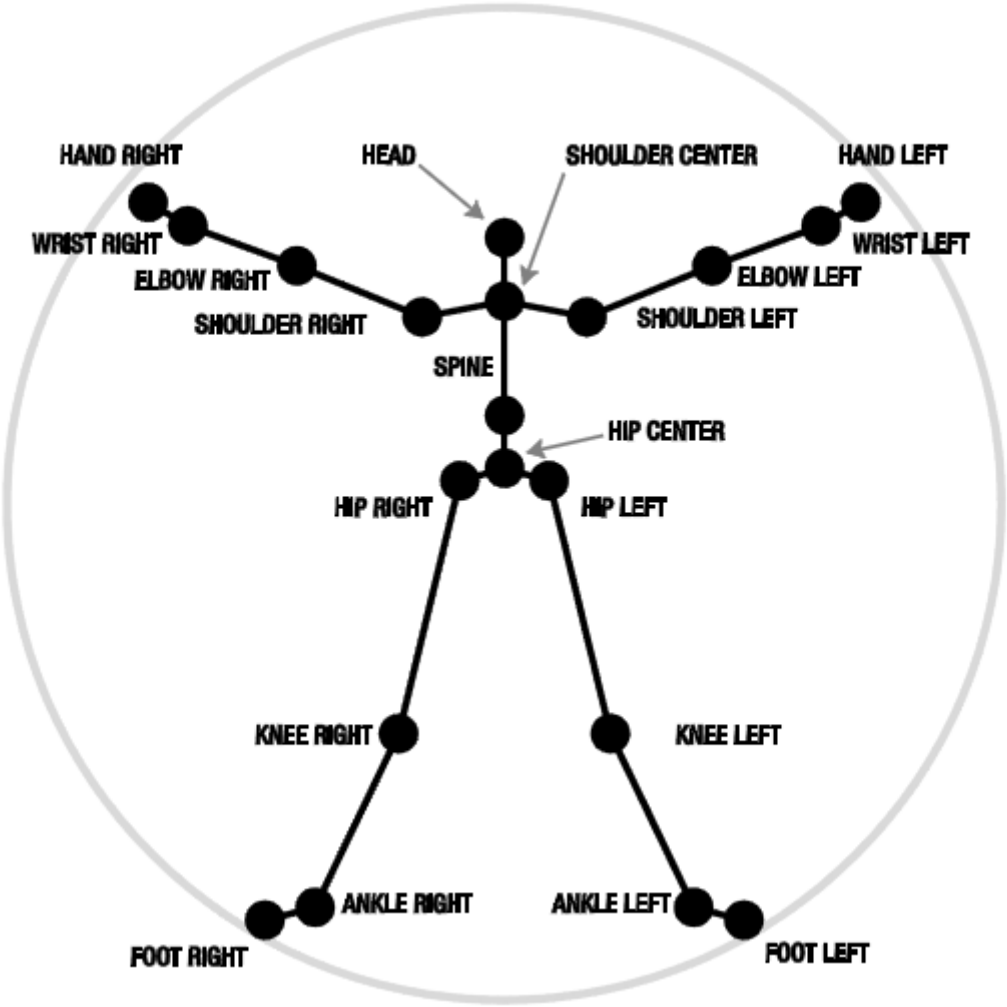


Figura 1.15.- Huesos del Esqueleto

Así, cuando el usuario se mueve a la izquierda, a la derecha o salta, el sensor capturará el movimiento y lo trasladará a la aplicación.

Figura 1.16.- Seguimiento esquemático

A su vez, este software es capaz de dar soporte para el reconocimiento facial y permitir, así, distinguir de manera individual el rostro de los usuarios que se acercan al periférico de la cámara RGB. Además, se posibilita recordar al jugador recopilando datos físicos que se almacenan en un perfil asociado al mismo. De ahí que, cuando el usuario quiera volver a hacer uso de la aplicación, se puede determinar quién es y podrá comenzar a utilizarla fácilmente en cualquier momento con datos ya calibrados.

Figura 1.17.- Reconocimiento Facial

1.3.3 ¿A qué parece apuntar el dispositivo?

En un principio el dispositivo comenzó como un accesorio más (aunque revolucionario) para poder interactuar con la consola de videojuegos Xbox360. Hoy en día, Kinect ha llegado mucho más lejos que eso y muchas aplicaciones desarrolladas en todo el mundo están siendo innovadoras con el uso de este pequeño gran dispositivo. Otras áreas que no sólo involucran el entretenimiento están haciendo uso de este poderoso sensor, en aplicaciones que pretenden a ayudar a niños con autismo, planteando nuevas formas de comunicación, por ejemplo.

2 Desarrollando el proyecto

2.1 Idea Principal

El objetivo principal del proyecto es poder conectar el dispositivo de Kinect de microsoft con el motor de juego Unity. Para ello se nos plantean varias cuestiones que intentaremos enumerar en este documento.

2.2 ¿Qué necesitamos?

Tenemos que tener básicamente, el dispositivo Kinect o compatible, los drivers para dicho dispositivo y Unity3d.

2.2.1 Dispositivos

Actualmente a la venta (en desarrollo hay unos cuantos más) podemos encontrar básicamente dos modelos de dispositivos de capturas de movimiento, que son el Xtion de Asus y la Kinect de Microsoft.

Figura 2.1.- Xtion de Asus y Kinect de Microsoft

Las características de ambos son similares, pues ambos tienen cámaras RGB, sensor de infrarojo y de profundidad. La diferencia radica en que la kinect tiene 4 micrófonos mientras que la Xtion básica tiene 2.

2.2.2 Drivers o controladores

Los dos controladores o drivers más utilizados para estos dispositivos son OpenNI y Microsoft Kinect SDK.

Figura 2.2.- Logos de las dos controladores

OpenNI lleva más tiempo en el mercado que microsoft Kinect SDK, por tanto hay muchos más ejemplos y comunidades en Internet. Incluso, hay un Wrapper[10] (o envoltorio) para usar la Kinect con Unity3D.

Sin embargo, la versión beta del controlador SDK de Microsoft salió en verano de 2011, pero la versión Release (o final) no la sacaron hasta el 1 de febrero de 2012. Lo que significa que hay muchos menos ejemplos que en la versión anterior, y además hay cambios "significativos" entre la versión beta y la versión definitiva.

2.2.3 ¿Qué camino elegir?

La decisión lógica hubiera sido: "OpenNI", ya que tiene más ejemplos, tiene una comunidad más grande y además tiene un Wrapper para Unity3D, lo necesario para el proyecto que se quiere plantear.

Después por contra, tenemos a Microsoft SDK, que sólo tenía muy pocos ejemplos en su versión beta, la definitiva estaba recién salida (prácticamente sin ejemplos) y no tenía oficialmente ningún Wrapper para Unity3d, como su competidora.

Después de mirar vídeos y webs sobre OpenNI, Microsoft SDK y Unity3D, me encontré un vídeo que se titula Kinect American Football Demo[11], en el cuál Andrew Devine de la Universidad Central de Florida[12] estaba realizando su

[10] http://en.wikipedia.org/wiki/Wrapper

[11] http://www.youtube.com/watch?feature=player_embedded&v=YWRQE06-pz4

[12] http://www.eecs.ucf.edu/isuelab/unity.php

proyecto sobre Microsoft SDK Beta, Kinect y Unity3D, con una peculiaridad que no había visto hasta ahora, le incrustó el reconocimiento de voz en Unity3d.

Figura 2.3.- American Football Demo with Kinect and Unity3d

Así que, a partir de aquí, mi mente dijo: Podemos intentar hacer un Wrapper que use Microsoft SDK 1.0 Release (ya que Kinect es de Microsoft), tiene el reconocimiento de audio y parece que es posible unirlo con Unity3D. Además, estaba convencido (como al final fue) de que Microsoft iba a seguir sacando nuevas versiones (tanto de su dispositivo como de sus drivers), para potenciar la kinect y juegos para la xbox 360.

2.2.4 ¿Cómo ensamblar las piezas?

Teniendo claro que empezaremos con Microsoft Kinect SDK, Unity3D y el editor o entorno de desarrollo Visual Studio (ya que es compatible con Unity3d), tenemos ahora que ver con qué lenguaje de programación escribiremos el código.

2.2.5 ¿Qué lenguaje escoger?

Con Unity3D, a parte de assets que nos da cosas hechas, podemos realizar nosotros mismos nuestro código. Este código en Unity3D lo catalogan como scripts[13], que son órdenes normalmente simples que se almacenan en un fichero

[13] http://es.wikipedia.org/wiki/Script

de texto. Para estos scripts, Unity3d soporta los lenguajes de programación de javascript, C# y boo, este último es un derivado del phyton.

Microsoft Kinect SDK nos ofrece la posibilidad de programar en C++, C# y en la última versión con Visual Basic. Asimismo, la última versión ya es compatible con su Framework[14] de videojuegos XNA[15]. Cabe destacar que con XNA podemos programar para la consola XBOX360, PC y Windows Phone pero todavía con la licencia actual no se puede usar la kinect con la Xbox360, solamente en PC.

Parece que la decisión está clara, usaremos el lenguaje C#, que es compatible tanto en Kinect con en Unity3D.

2.2.6 "Complicaciones" con el lenguaje elegido

Cuando empiezas a programar, entender y ver cómo hacer las cosas, te das cuenta de que no resulta tan fácil usar el lenguaje C# dentro de Unity3D para conectar la Kinect. La razón estriba básicamente en el .NET Framework[16]. Este Framework es un componente de software incluido en los sistemas operativos Microsoft Windows que provee soluciones pre-codificadas para requerimientos comunes de los programas y gestiona la ejecución de programas escritos específicamente para este framework.

Figura 2.4.- Logos de .NET y Mono

Microsoft Kinect requiere una versión de .NET Framework igual o superior a la 4.0 para poder funcionar mientras que Unity3D trabaja nativamente con la versión 2.0 aunque puede llegar a trabajar con la 3.5. Leyendo y buscando en foros de Unity3d, observamos que el .Net Framework que usa Unity3d se basa en Mono[17], que es un desarrollo aparte de microsoft pero con la misma idea. Este desarrollo[18] no es 100% compatible con el framework 4.0 por el momento.

[14] http://es.wikipedia.org/wiki/Framework
[15] http://es.wikipedia.org/wiki/XNA
[16] http://en.wikipedia.org/wiki/.NET_Framework
[17] http://www.mono-project.com/Main_Page
[18] http://www.mono-project.com/Compatibility

2.2.3 Envoltorio y ayudas

Investigando más sobre Unity3D, podemos ver que acepta plugins [19] externos. Un plugin o complemento es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

Por tanto, la idea básica consiste en hacer un plugin o dll intermedia que conecte el código de unity3d con el código de la Kinect.

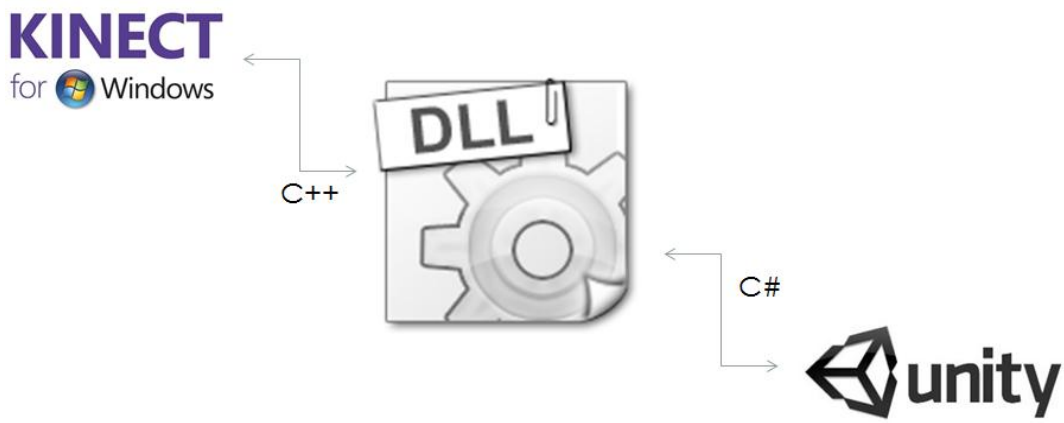


Figura 2.5.- Diagrama de conexión entre Unity y Kinect

El plugin o dll puede estar programado en C++ o C#. Después desde C# en Unity3D, podemos llamar a dicha dll. Los pocos ejemplos que había en su comienzo para entender el funcionamiento de Kinect están hechos en C#, ya que es un lenguaje más "fácil" de usar y más intuitivo que C++.

En codeproject[20] hay una buena explicación de cómo hacer librerías en C++ y C#, así que opté en hacer una librería en C# (que se convierte en una dll), para cargarlo en otro C# desde Unity3D. La idea es muy buena hasta que te dice Unity3D: "no puedo cargar esta librería dll porque está hecho en el framework 4.0" (que es el necesario para Kinect y Unity no lee, debí imaginarlo, pero no perdía nada en probarlo).

Figura 2.6.- Logo de The Code Project

[19] http://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica)
[20] http://www.codeproject.com/

Por tanto, hay que intentar entender el único ejemplo que había en ese momento en C++ de Kinect, y buscando en Internet, nuevamente, y gracias a Andrew Devine[21] citado anteriormente (y que recientemente en mayo publicó parte de su código), pude comprender gran parte de lo que estaba escrito en C++. Por tanto, se consiguió hacer una librería en C++ (ya no hace falta el .NET Framework, porque el C++ es un lenguaje de más bajo nivel), y desde Unity3D con C# se pudo leer la dll (programada en C++) "sin problemas".

El problema vino en la parte del audio, que no había ningún ejemplo del reconocimiento de voz en C++, sólo en C#. Entonces, esto me llevó a usar doble pasarela, es decir a realizar dos procesos de conversión. Tenemos una dll programada en C# de audio, esta librería la lee nuestra librería en C++ (que hace de puente). Después nuestro código en Unity sólo pregunta a la librería C++.

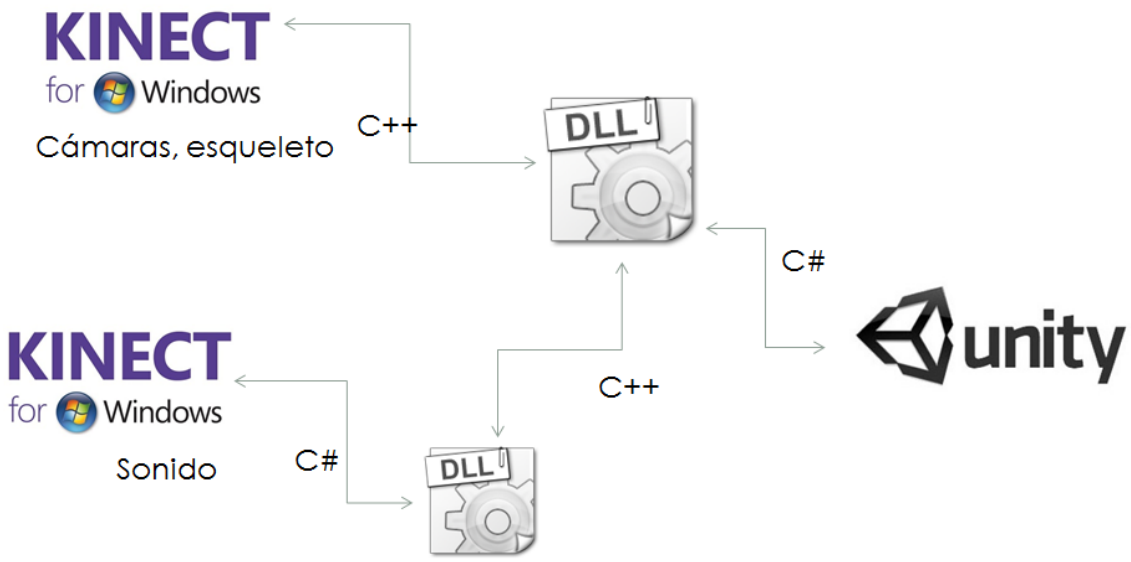


Figura 2.7.- Diagrama de conexión entre Unity y Kinect II

Un proyecto a destacar realizado con Kinect y Unity es una wiki que se encuentra alojada en Entertaiment Technology Center Carnegie Mellon [22], aunque está realizado con la versión SDK de Microsoft, está integrada en Unity3D en C++.

Un libro que me ayudó a comprender más kinect es el único libro que hay sobre Microsoft Kinect SDK y es "Beginning Kinect Programming With the

[21] https://github.com/adevine1618/KinectSDK-Unity3D_Interface_Plugin

[22] http://wiki.etc.cmu.edu/unity3d/index.php/Microsoft_Kinect_-_Microsoft_SDK

Microsoft Kinect Sdk"[23] que adquirí por Internet publicado en marzo. Me llegó dos meses después de que saliera la Release 1.0.

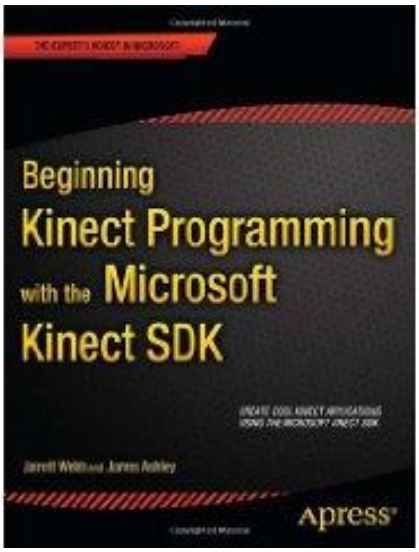


Figura 2.8.- Portada del libro

Un proyecto bastante bueno y que hay que tener en cuenta, desde mi humilde opinión, es el de Zigfu[24], una empresa que tenía en su idea también un wrapper o envoltorio para la kinect y Unity3d realizado con OpenNI. Después de la versión 1.0 de Microsoft Kinect SDK, hicieron su Wrapper para que fuera compatible con la misma y OpenNI. Aunque, actualmente, está en fase beta, se puede probar con una marca de agua o comprarla. El problema es que por ahora no tiene reconocimiento de voz. Y por tanto, tuve que realizar un Wapper que reconociera comandos de voz.

[23] http://www.amazon.es/Beginning-Kinect-Programming-Microsoft-SDK/dp/1430241047
[24] http://zigfu.com/

3. El Proyecto

3.1 ¿Qué necesitamos antes de empezar?

Necesitamos una Kinect, el Unity3D (a menos que nos dieran el ejecutable para probar el resultado final), Microsoft Kinect SDK (la última versión a día de hoy es la 1.5). Esto es lo básico y fundamental para poder ejecutar el proyecto en Windows 7.

3.2 El proyecto en Unity3d

Al abrir el proyecto, lo primero que vemos es un escenario en el cual vemos un paisaje con un perro de raza schnauzer. Tengo que agradecer a Juan Antonio González[25] quien se encargó de realizar todo el diseño, texturizado y animación del proyecto, pues sin él no lo hubiera acabado "tan rápido" (¡gracias Juan!).

Figura 3.1.- Captura inicial del proyecto sin ejecutar

[25] http://www.youtube.com/watch?v=doaz-2S96S4

3.2.1 Contenido en la ventana Project de Unity3d

El proyecto está dividido en carpetas o directorios. En la carpeta Scripts podemos encontrar el código que hace falta para ejecutar el proyecto, y es en esta carpeta donde más nos centraremos a explicar en este apartado.

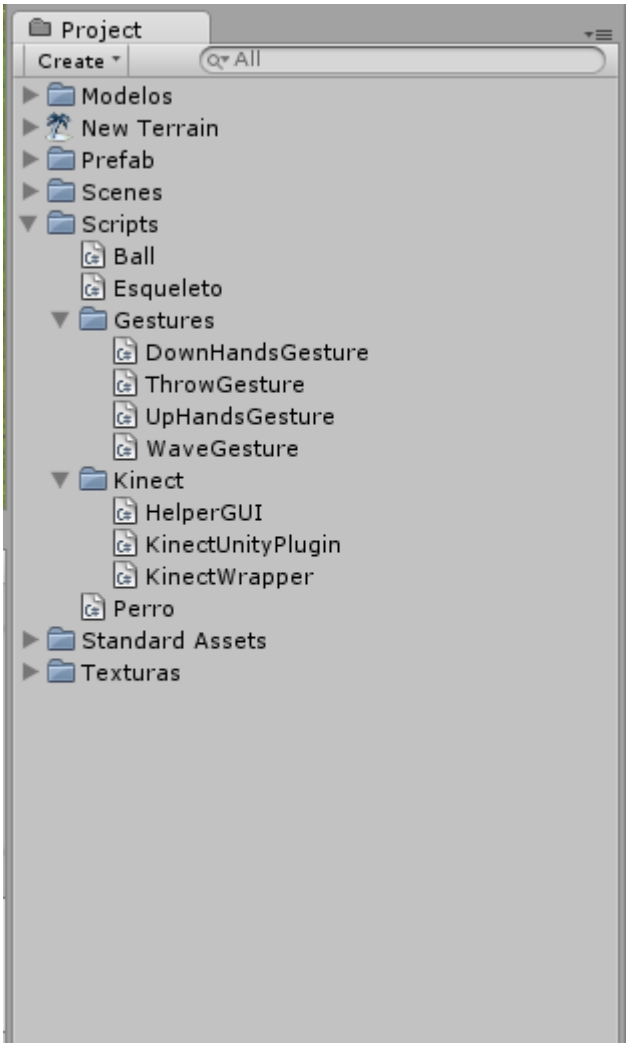


Figura 3.2.- Panel Project en Unity

Modelos: Contiene los modelos que usamos en el proyecto, como por ejemplo el perro.

New Terrain: Es el suelo y árboles que se ven.

Prefab: Es una característica de Unity3D en la cuál podemos tener objetos o elementos prefabricados para poder usarlos en cualquier momento. Lo que se

hace es colocar una instancia del objeto en la escena, idéntica como está grabada incluyendo código, colores, materiales, etc.

Scenes: Son las escenas (o niveles o pantallas) que tienen el juego. En este caso sólo tenemos la principal.

Scripts: Lo comentaremos en el siguiente apartado.

Standard Assets: Son assets que trae el Unity3d y hemos cargado para usar. Por ejemplo en este caso usamos el cielo, el terreno y los árboles.

Texturas: Aquí tenemos almacenadas la imágenes y materiales que se usan en el juego.

Scripts

Como hemos dicho anteriormente, los scripts pueden estar escritos en javascript, C# o boo. En nuestro caso, hemos usado el lenguaje C#, ya que probablemente es el más completo de los tres aunque probablemente el javascript sea más usado en Unity3D por su facilidad de uso.

Los scripts están divididos en 3: Escena, Gestures y Kinect.

Escena

En la raíz del directorio, encontramos los códigos necesarios para el funcionamiento del "juego", por ejemplo la Ball que irá enganchada al objeto Ball. El Perro tiene características y animaciones especiales que tiene que hacer dependiendo de lo que deseamos.

Gestoures

Aquí tenemos la programación de los gestos que hará el personaje. Obtenemos los datos que nos devuelve la kinect, los procesamos y vemos si los movimientos coinciden con algo que deseamos.

WaveGesture es un código que nos da, en este caso, si el personaje está saludando con la mano derecha por ejemplo. Tenemos varios gestos programados como Levantar las dos manos, bajar ambas manos y lanzar.

Kinect

Aquí encontramos el núcleo de la conexión entre los sistemas (unity y kinect).

- **KinectWrapper.cs** es donde tenemos la conexión básica con nuestra dll. Podemos cambiar la ruta de la dll en : const string msKinectUnityDLL = "KinectUnityPlugin.dll". Además, podemos encontrar los Joints que mostramos anteriormente en una figura y las llamadas a la librería.

- **KinectUnityPlugin.cs** es la parte visible en Unity3d. Ésta la conectaremos a la cámara principal (main camera), que explicaremos después, pero prácticamente tenemos la configuración básica de la escena. En la parte Star() tenemos comentado un código (y que no me pareció necesario colocarlo como opción) que es mover el motor de la kinect. Con KinectWrapper. SetCameraAngle(-20); movemos el angulo de inclinación de del dispositivo -20 grados. El dispositivo puede oscilar entre -27 y 27 grados, según la documentación.

- **HelperGUI.cs**. Es una ayuda que nos sirve básicamente para dibujar líneas en Unity como la simulación del esqueleto en pantalla

3.3.2 Contenido de la ventana Hierarchy

Este panel contiene lo que vemos en el juego que serían los modelos, luces, cámaras, terrenos, etc. Comentaremos los principales para el proyecto.

Figura 3.3.- Panel de Herarquía del Unity

Main Camera

Esta cámara es lo que el jugador ve cuando juega, aquí encontramos un script fundamental para enlazar Unity y Kinect, cuyo nombre es KinectUnityPlugin.cs

KinectUnityPlugin.cs

Aquí tenemos una serie de valores que activaremos o desactivaremos según nuestras necesidades.

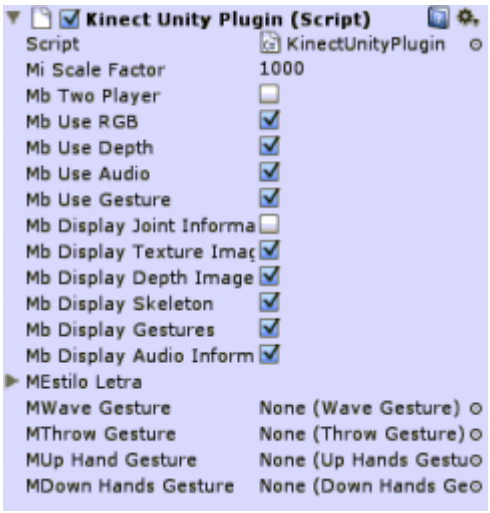


Figura 3.4.- Script KinectUnitPlugin.cs

El factor de escala es para compensar los valores entre Unity y kinect, en los cálculos.

Los dos player no está del todo implementado.

A continuación, podemos utilizar si lo deseamos las siguientes opciones de la kinect: la Cámara RGB, la cámara de Profundidad, Audio y Gestos.

Los bloques de display son de textos o gráficos para mostrar la información recibida en la pantalla del Unity.

Pantalla Principal del juego

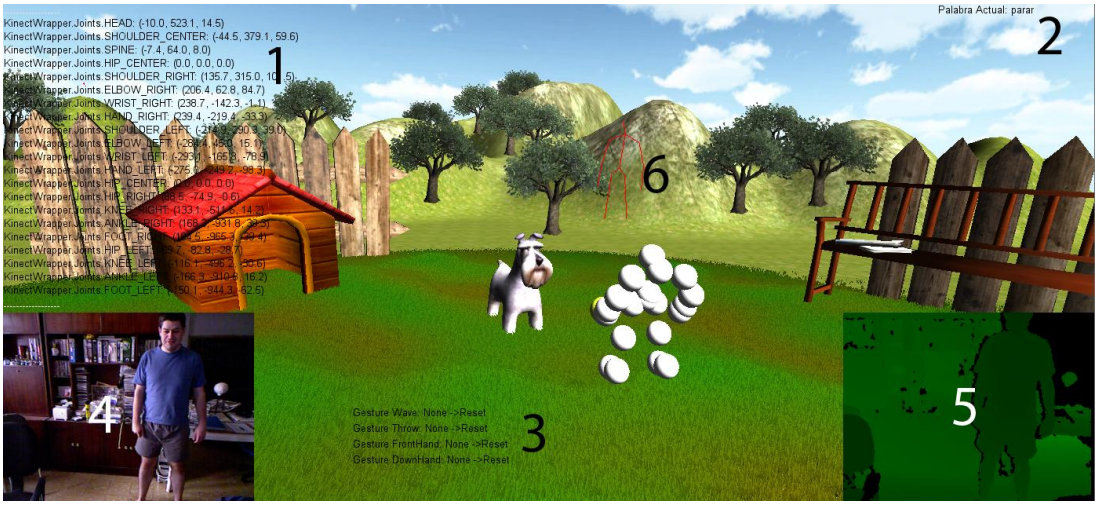


Figura 3.5.- Partes del juego

1. Muestra la información del skeleton
2. La palabra actual que reconoció la kinect
3. Información de los gestos recibidos
4. Cámara RGB
5. Cámara de profundidad
6. Representación del esqueleto en 2D

3.4 ¿Qué podemos hacer?

Aquí podemos ver el resultado final del proyecto. En la imagen podemos ver elemento de la escena, tenemos a un perro, un personaje representado con 20 bolas que forman la parte básica del Skeleton de Kinect y una pelota de tenis. Todo lo demás es decorado.

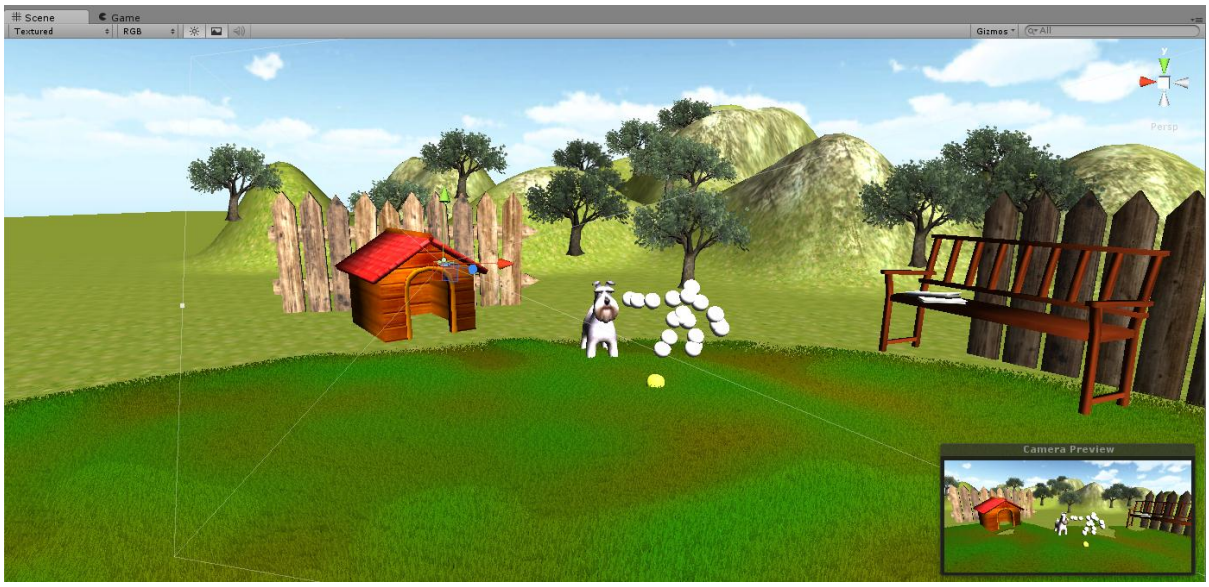


Figura 3.6.- Captura de una imagen del juego

Podemos hacer movimientos y los veremos reflejados en nuestro esqueleto en tiempo real. También podemos dar diferentes órdenes de voz o con gestos, que pasaremos a detallar.

3.4.1 Órdenes de voz

A continuación, se mostrarán con captura de imágenes lo que hace el juego a partir de una palabra u orden que le decimos a la kinect y el Unity recoge y procesa.

Siéntate

Al decir esto, el perro se sienta, como en la imagen.

Figura 3.7.- Perro sentado

Levántate

Figura 3.8.- Perro de pie

Baila

Figura 3.9.- Perro bailando

3.4.2 Órdendes por Gestos

Se han implementado varias órdenes por gestos que hacemos.

Wave (Saludar)

Esto lo hacemos con la mano derecha (por ahora sólo con esa) y si lo hacemos el perro salta.

Figura 3.10.- Gesto de saludar y perro saltando

Manos arriba (UpHands)

Si levantamos las dos manos hacia arriba, por encima de la cabeza, el perro se hace el muerto.

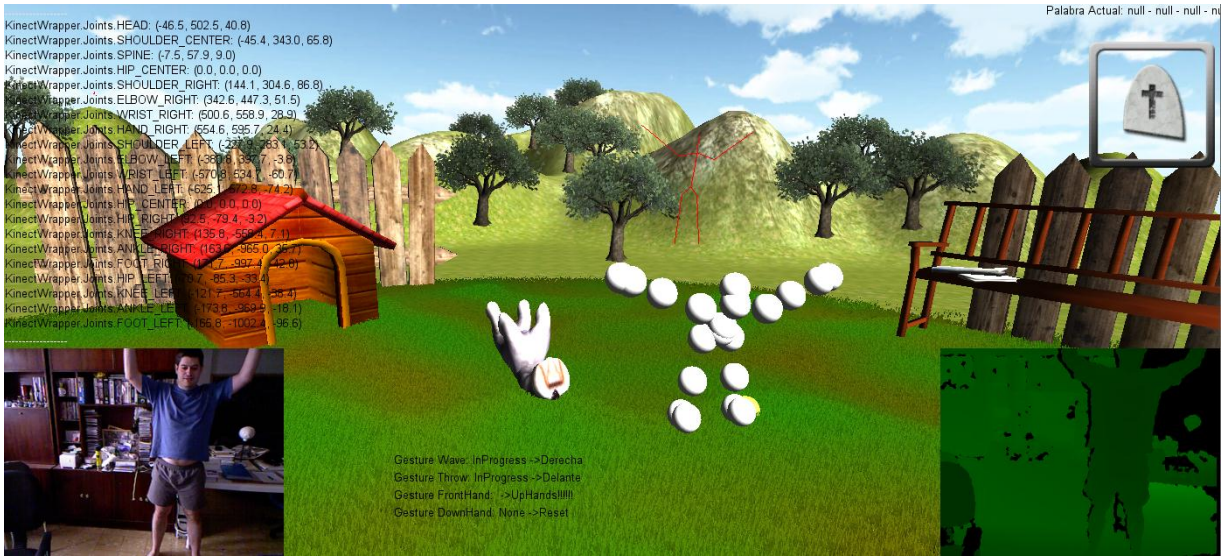


Figura 3.11.- Manos arriba (perro haciéndose el muerto)

Manos abajo (downHand)

Si nos agachamos y ambas manos están por debajo de la rodilla, el esqueleto coge la pelota del escenario.

Figura 3.12.- Manos abajo

Lanzar

Si tenemos la pelota en la mano, lo que hacemos es poner la mano hacia atrás y después hacia delante y la pelota saldrá disparada.

Comportamiento del juego

- El perro está estático mientras no digamos lo contrario.

- El esqueleto se mueve y sigue nuestros movimientos.

- El perro tiene animaciones "estáticas" que se van repitiendo. Por ejemplo, mueve la cabeza y la cola mientras espera órdenes.

- Un segundo después de lanzar la pelota el perro irá a buscar la pelota y nos la traerá. Tiene una inteligencia artificial muy básica. Va al punto donde cae la pelota y la trae al punto del esqueleto. Para esto por ejemplo usamos vectores y buscamos la distancia entre dos puntos, aunque Unity nos ayuda mucho en esto.

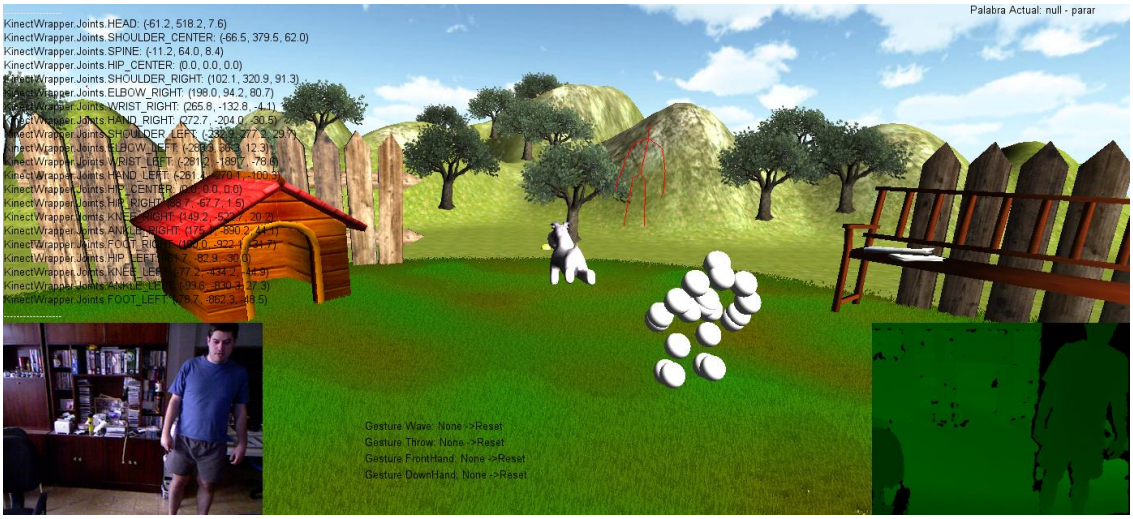


Figura 3.13.- Perro corriendo

3.5 Por hacer y mejorar (ToDo)

3.5.1 Sonido

Con la nueva actualización de la Kinect (finales de mayo a 1.5), aparecieron más ejemplos de ayuda para obtener el sonido a través de c++, antes no existía ninguno. Ahora en lugar de hacerlo con un plugin c#, se puede ver los ejemplos e intentar hacerlo en c++.

3.5.2 Sonido 2

Ahora mismo no se pueden añadir palabras al diccionario desde Unity3D, sino las que tengo escritas y grabadas en el plugin de C#. No resultaría muy difícil añadir nuevas palabras desde Unity3D en un script.

3.5.3 Sonido 3

En estos momentos, sólo reconoce palabras sueltas: "levántate", "siéntate",...no reconoce frases como "hazte el muerto", sería un punto interesante a tener en cuenta.

3.5.4 Sonido 4

Aunque está actualizado para interpretar palabras en español (con la actualización 1.5), hay ciertas palabras que por mucho que se las incluya no las acepta, habrá que averiguar por qué sucede esto. Un ejemplo es "muérete" nunca la acepta y está escrita. Y "pelota" está pero la acepta muy de vez en cuando.

3.5.5 Rotación

El plugin ahora mismo sólo detecta posiciones en el espacio tridimensional (w, y, z), pero no la rotación, por ejemplo girar la cara o la mano. Tenemos un modelo de un niño pero al no rotar queda "raro". Por eso, al final en nuestro proyecto quedó el esqueleto de esferas como representación humana.

3.5.6 Posiciones

Al principio, estaba perdido en cómo pasar las posiciones del espacio de kinect al espacio de Unity3D, entonces tengo conversiones que funcionan pero estando la escena en determinada posición. Ahora que tengo más idea se puede "mejorar" el sistema, para una futura versión.

3.5.7 Colisiones

No tenemos colisiones con el escenario, pero realmente no es importante, y en Unity3D es fácil de colocar.

3.5.8 Modo NEAR

El modo NEAR no está implementado en el actual proyecto pero puede suponer una mejora en la funcionalidad de la Kinect y no es complicado de utilizar. De todas formas, todavía no está a la venta el dispositivo que acepte este modo. Consiste en poder detectar elementos a 40 centímetros de distancia como mínimo.

3.5.9 Dos jugadores

En relación a este aspecto hay que decir que el código fuente no tiene implementada la opción de dos jugadores, de ahí que en el presente trabajo, sólo está activada la opción de un único jugador. No obstante, creo que no resultaría difícil asignar otro player.

3.5.10 Gestos (Gestoures)

Hay varios gestos hechos. En Unity está el valor de usar o no todos los gestos. Realmente, se podría personalizar para usar determinados gestos dependiendo de la situación o escena.

4. Conclusiones

La verdad es que este proyecto fue gratificante pero muchas veces frustrante. La razón principal es que apenas había información sobre Microsoft Kinect SDK 1 como ya comenté, ya que salió a la vez que cuando empecé con el proyecto. Por tanto, no tenía mucha documentación.

Depurar el código en Unity con Visual Studio y una librería de terceros es muy complicado, ya que por ejemplo hay veces que el Unity se cierra inesperadamente sin dejarte información (tras un fallo). Por tanto, tenía que ir con pies de plomos y muy despacio. Además, se trabajó con dos librerías, se compila una (C#), se compila la de C++ (que tira de la anterior), y después se ejecuta en Unity. Antes de compilar esas librerías hay que cerrar el Unity para que libere la librería (que tiene en ejecución y no permite reescribir el fichero dll) para poder compilar la nueva.

Aunque quedan cosas por hacer, el resultado final me gustó bastante. No obstante, he de decir que me hubiera gustado mejorar algunos aspectos del proyecto, si hubiera más tiempo, quizás más adelante lo haga y lo publique en www.polyfactor.com[26] (si es posible hacerlo y tengo tiempo, claro).

Este proyecto está realizado principalmente con: Unity3D, Microsoft Kinect y Microsoft Kinect SDK, Visual Studio 2010 y Blender.

Cabe destacar un proyecto que se llama Leap Motion[27], que es un sistema de control gestual mucho más pequeño y barato que la Kinect. Todavía no está a la venta pero podríamos estar en el comienzo de verdad de controlar todo con gestos.

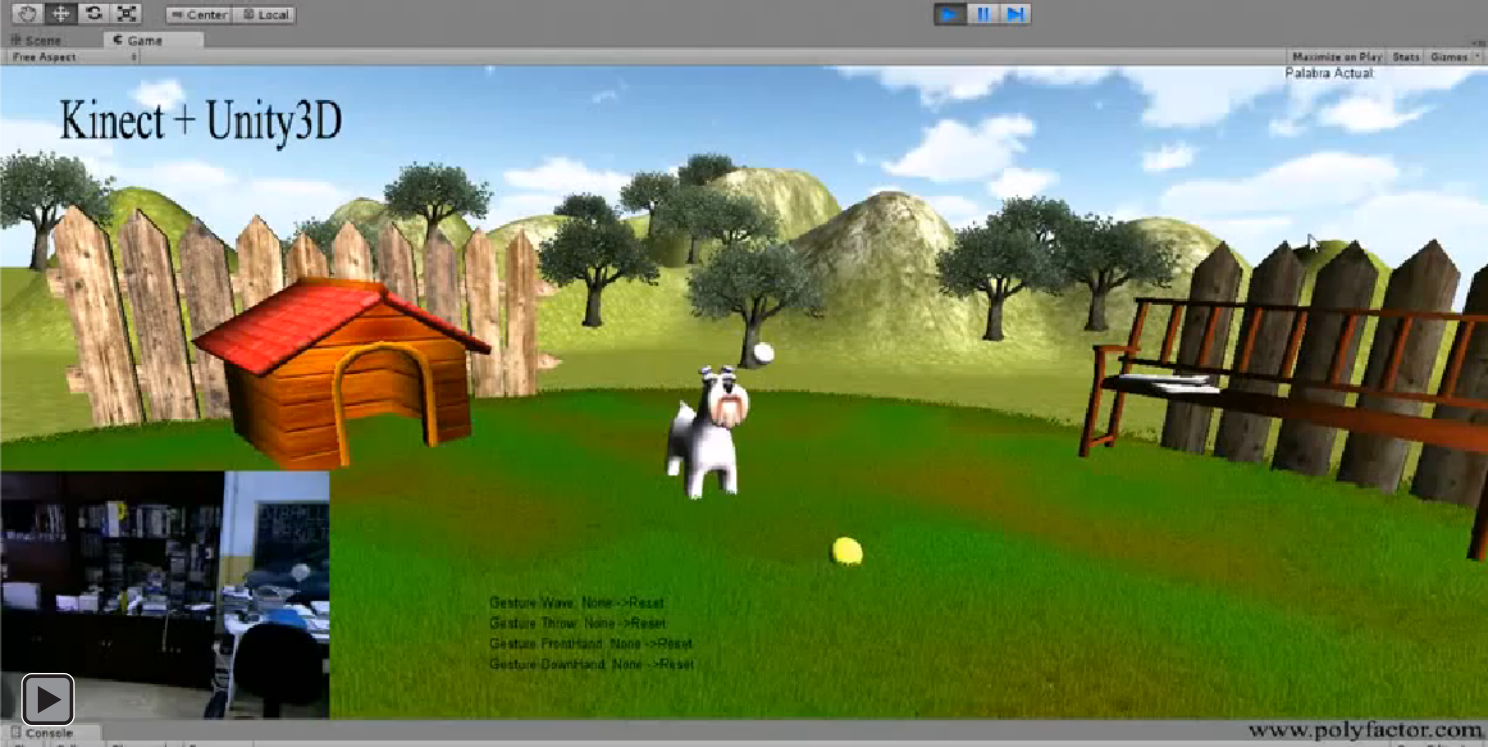
No quiero dar por concluido este informe sin agradecer a todos los que he mencionado a lo largo de este documento en sus correspondientes categorías, ya sean personas o páginas webs, a Noelia Campos por releer y corregir este documento y al tutor Don Fernando Fraile por sus directrices y ánimos a largo de todo el proceso. ¡Muchas gracias!

[26] http:// www.polyfactor.com
[27] http://www.leapmotion.com/

5. Resultado Final en Vídeo[28]

(vídeo incrustado en el pdf entre esta página y la siguiente)
(enlace al vídeo en la nota al pie de página)

[28] http://www.youtube.com/watch?v=9KwAdsV8PE4


Center
Local
Scene
Game
Free Aspect
Maximize on Play
Stats
Gizmos
Palabra Actual:
Kinect + Unity3D
Gesture Wave: None ->Reset
Gesture Throw: None ->Reset
Gesture FrontHand: None ->Reset
Gesture DownHand: None ->Reset
Console
www.polyfactor.com

6. Lista de referencias

Libros de consulta:

Beginning Kinect Programming with the Microsoft Kinect SDK. Jarrett Webb and James Ashley. Apress