E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Título del Trabajo Fin de Grado

Grado en Ingeniería Informática

Ignacio Carmelo Oteiza Solchaga

Benoit Pierre Bossavit

Pamplona, 18 de junio de 2018

Resumen:

The name of this project is *BrickUp3D*. It is a game which is about building structures, like towers or castles for example, using your own hands as the controller.

The game is made to be played by children about 4 years old. It aims to simulate a real behavior by playing with figures like cubes or cylinders, where the goal is to build a specific structure carefully, without make it fall. So, like in the real life, the player can just raise their hands and a representation will appear in the screen and will reproduce hands' movement pushing the figures like if it were their real hands. In addition, if the player closes a hand when it is near a figure, it will grab that figure and carry it wherever the player wants. To drop the figure the player has to open the hand.

The game has different levels with different difficulty, but it provides some helpful mechanics such as magnet or glue to make it easier for children, so they can have a good time while playing.

Índice

1.- Introducción:

Este trabajo responde al nombre de *BrickUp3D* y se trata de un juego serio dirigido a niños de unos 4 años, que se basa en apilar piezas formando un castillo o construcción controlando unas manos virtuales como si fueran las propias, mediante el uso de la kinect2.

1.1.- Objetivos:

El objetivo de este proyecto es poner a prueba la capacidad del usuario para interactuar con la física de objetos con distintas formas y tamaños al juntarse en distintas estructuras con dificultad variable.

Permitirá mediante un juego entretenido para los niños evaluar por un lado su coordinación y por otro su capacidad cognitiva y de percibir el entorno. Además, en caso de tener un exceso de dificultades a la hora de resolver algunos de los niveles, prever la posibilidad de que se desarrollen ciertas enfermedades en el futuro como por ejemplo el autismo.

1.2.- Herramientas utilizadas:

Dividimos las herramientas utilizadas en Hardware y Software.

1.2.1.- Hardware:

- **Kinect V2:**

La Kinect es un controlador de juego desarrollado por Microsoft que permite a los usuarios controlar e interactuar con la máquina sin necesidad de tener un controlador de videojuegos tradicional, como un mando o el ratón y teclado, reconociendo los gestos, comandos de voz y objetos e imágenes.

He utilizado la *Kinect2 for Windows* y se ha empleado para controlar el cursor de selección de niveles y para manejar las manos virtuales con las que se realizan las construcciones necesarias para completar los niveles.

1.2.2.- Software

Las herramientas Software las dividimos en programas y lenguajes de programación.

1.2.2.1.- Programas:

- **Unity:**

Unity es un motor de videojuego multiplataforma creado por Unity Technologies.

He empleado Unity para crear y dar características a todos los elementos del juego. También para realizar los diseños de los interfaces y sus animaciones.

- **Microsoft Visual Studio:**

Microsoft Visual Studio es un IDE (entorno de desarrollo integrado) para sistemas operativos Windows. Soporta múltiples lenguajes de programación como C++, C#, Visual Basic, .Net, F#, Java, Python, Ruby y PHP.

Me he servido de este entorno para programar en C# el comportamiento y funcionamiento tanto de las figuras entre sí como su iteración con las manos que maneja el usuario.

- **Blender:**

Blender es un programa informático multi-plataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. Es compatible entre otros con Windows, Mac OS X y GNU/Linux.

Este programa ha sido empleado para modelar alguna de las figuras básicas del proyecto en 3D.

1.2.2.2.- Lenguajes:

- **C#:**

Para controlar los objetos en Unity se debe de crear scripts que indican las instrucciones que deben seguir asignandoselos a cada uno. Estos scripts con instrucciones pueden estar escritos en JavaScript o C#. En este caso están escritos en el lenguaje de programación C#.

1.3.- Plataforma:

Este proyecto ha sido pensado para ser ejecutado principalmente en un ordenador con Windows 8 y Windows 10. Sin embargo, si se cambiara el sistema de reconocimiento del cuerpo para controlar el juego sería sencillo hacer que funcione para otros sistemas operativos para los que no funciona la Kinect ya que Unity permite cambiar la plataforma de salida de manera muy sencilla. En el caso de Linux ni siquiera haría falta cambiar la Kinect porque también es compatible.

Por otro lado también podría emplearse en la Xbox, sin cambiar la Kinect por otro dispositivo, realizando modificaciones mínimas. Simplemente debería añadirse la capacidad de navegar por el menú inicial, del que hablaremos más tarde, con un mando de Xbox y Unity ya podría generar el juego para esta consola con las mismas características que el de PC.

1.4.- SetUp:

- Kinect V2 for Windows:

- Portatil Lenovo MT 20354 con procesador Intel® Core™ i7-4500U CPU @ 1.80GHz y tarjeta gráfica GeForce 840M

1.5.- Estado del arte:

Se han desarrollado muchos proyectos con Kinect desde que salió en 2010. Entre ellos podemos encontrar algunos con objetivos o apariencias similares al de este proyecto y que utilizan además la Kinect como herramienta.

1.5.1.- Stroke Recovery with Kinect:

Stroke Recovery with Kinect[1] es un proyecto desarrollado por Microsoft dirigido a personas que habían sufrido un accidente y tenían que realizar ejercicios de rehabilitación.

Como estos ejercicios suelen ser sencillos, monótonos y aburridos lo que proponía era una serie de juegos serios con los que los pacientes podrían realizar la

rehabilitación de manera más amena. Además, los pacientes podrían realizar estos ejercicios en su casa.

Estos ejercicios son los 3 siguientes:

- Copiar la pose de la pantalla
- Mover cubos de un compartimento a otro
- Controlar una nave con la mano

De estos 3 ejercicios el que más parecido guarda con el proyecto es el segundo, sin embargo, únicamente consiste en desplazar las figuras de un lado a otro, mientras que *BrickUp3D* consiste en moverlas y juntarlas entre sí para formar construcciones, por lo que se requiere una mayor precisión y percepción del entorno y las físicas. Además, no tiene esa apariencia de ejercicio porque no tiene que realizarse constantemente el mismo movimiento de pasar un cubo de un lado a otro y eso lo hace más atractivo para los niños, que son los usuarios a los que está dirigido este proyecto.

1.5.1.- Controlar robots con la Kinect:

Por internet hay varios proyectos relacionados con controlar un robot con la Kinect usando el cuerpo. Uno en específico, en el que según la fuente ha formado parte la NASA, implementa también realidad virtual para llevarlo a cabo.

Su proyecto[2] consiste en manejar el brazo de un robot como si fuera el de uno propio. Desde el Oculus Rift, que son las gafas de realidad virtual, se proyecta un escenario como si fuera lo que se ve desde el punto de vista del robot. Entonces el usuario mueve su brazo y la Kinect detecta sus gestos y se lo manda al robot, que mediante algoritmos de inteligencia artificial lleva a cabo estos movimientos.

El objetivo es poder mover un robot desde marte como si el usuario fuera el que estuviera allí. Para realizar pruebas se intenta coger un objeto de una mesa mientras el usuario permanece a distancia manejando el robot de la manera que se ha descrito anteriormente.

Este proyecto tiene una mecánica similar a *BrickUp3D* en cuanto a que deben moverse objetos con una mano que se controla por medio de la Kinect. Sin embargo, difiere en que el objetivo es el de mover un brazo robótico mientras que en este proyecto el objetivo es evaluar las capacidades del usuario mientras juega y se entretiene.

[1] https://www.microsoft.com/en-us/research/blog/stroke-recovery-gets-a-boost-from-kinect/

[2] http://www.ign.com/articles/2013/12/31/nasa-uses-kinect-and-oculus-rift-to-control-a-robotic-arm

2.- Metodología empleada:

El proyecto ha sido dividido en varias etapas priorizando la funcionalidad principal para el orden en el que se han llevado las mismas. Se ha dividido de la siguiente manera. 1 día = 8 horas.

Etapas	Sub-etapas	Tiempo aproximado
Pantalla principal	Documentación inicial y establecimiento de entorno de trabajo	2 días
	Diseño para funcionamiento básico	8 días
	Implementación de ayudas	8 días
	Mejoras en diseño y apariencia	2 días
Menús de navegación	Creación de menú inicial	1 día
	Creación de menú de configuración	2 días
	Creación de selector de niveles	2 días
Pruebas y pequeños retoques	Varias pruebas y ejecuciones para comprobar el correcto funcionamiento.	3 días

2.1.- Documentación inicial y establecimiento de entorno de trabajo:

El primer paso para poder realizar el proyecto fue documentarse sobre la Kinect y averiguar cómo acceder a sus prestaciones con Unity.

En la API de la Kinect se pueden ver una serie de funcionalidades que ofrece la misma. Estas funcionalidades son:

- Reconocer y seguir el movimiento de la gente usando el seguimiento de su esqueleto.
- Determinar la distancia entre un objeto y el sensor de la cámara usando datos de profundidad.
- Capturar audio o encontrar la localización de la fuente del audio.
- Habilitar aplicaciones que se activan con la voz programando una gramática para usarla en una herramienta de reconocimiento de habla.

Pude probar algunas de estas funcionalidades con un programa que se instala junto con los drivers de la Kinect y permitía hacerse una idea de sus distintas posibilidades y capacidades.

De estas funcionalidades la que era más interesante para este proyecto era la de reconocer el movimiento de la gente usando su esqueleto. La API además ofrece ejemplos del uso de las herramientas de la Kinect para poder aprender a usarlas adecuadamente.

Para poder mover las manos se debía de hacer "Body Tracking". En la API se explica que la clase *BodyFrame*, a la que accede el *BodyFrameReader*, da acceso individual a los cuerpos mediante el método *GetAndRefreshBodyData.* Este método coge un vector de *Bodys* y actualiza las instancias de estos *Bodys* con los valores del *BodyFrame* actual. Un *Body* es un tipo que representa todos los datos que capta la Kinect de una persona.

En este caso solo se utilizará un *Body* a la vez ya que no se requiere ningún tipo de cooperación. Una vez que el *Body* tenga los valores asignados se puede acceder a los *Joints* de ese *Body* mediante el atributo *Joints* que devuelve un *Dictionary* con clave

JointType y valor del objeto que representa a ese *Joint* y que contiene toda la información necesaria. La Kinect posee también un listado de *JointType* al que se puede acceder mediante la instrucción: *Kinect.JointType.NombreDelTipo.*

Los *Joints* son las articulaciones del jugador representado con ese *Body,* pero para este proyecto solo se necesita obtener la información de las articulaciones que representan las manos, aunque la Kinect calcule todas. Por ello cada vez que se llama al método *RefreshBodyObject* se necesitaría coger únicamente la información que está en *Body.Joints* de los *Joints* de tipo *HandRight* y *HandLeft.*

A partir de estos *Joints* ya se podía obtener la posición de las manos, de manera que solo se le debería asignar a un objeto que representara a cada mano. Para simplificar el trabajo de relacionar la posición que la Kinect obtiene de la mano y la posición de la mano en la escena de Unity se centra el escenario y las figuras en la posición del mundo que devuelve la Kinect cuando se está a una distancia cómoda para jugar. Además, se normalizan los valores para que la mano no se desplace ni demasiada distancia ni demasiado poca y se multiplica por -1 el valor de la posición en el eje Z debido a que la Kinect está pensada para que el movimiento sea como un espejo, pero en esta ocasión la intención es que se muevan las manos como si fueran las del jugador.

También se puede obtener el estado de cada mano a partir del *Body*. Mediante la instrucción *Body*.*HandRightState* para el de la mano derecha y con *Body.HandLeftState* para el de la izquierda. Los posibles estados en los que se puede encontrar la mano están definidos en el enumerate *Kinect.HandState* y son los siguientes:

- Closed: La mano está cerrada.
- Open: La mano se encuentra abierta.
- Unknown: La Kinect desconoce el estado de la mano.
- NotTracked: El estado de la mano no se está siguiendo.
- Lasso: La mano está en un estado en el que tiene algún dedo bajado y alguno levantado.

2.2.- Diagrama de casos de uso:

Se decidió que el juego debía tener un panel de configuración en el cual el usuario pudiera cambiar la mano con la que mejor se maneja para navegar por el menú, también se aprovechó para incluir una opción de personalización para la mano 3d. Por otro lado, el usuario tiene la posibilidad de elegir un nivel y una vez escogido puede coger y dejar objetos, que son figuras geométricas sencillas, y montar estructuras con ellos sirviéndose de distintas ayudas como el imán y el pegamento. Cuando el nivel se completa se le informa automáticamente al usuario.

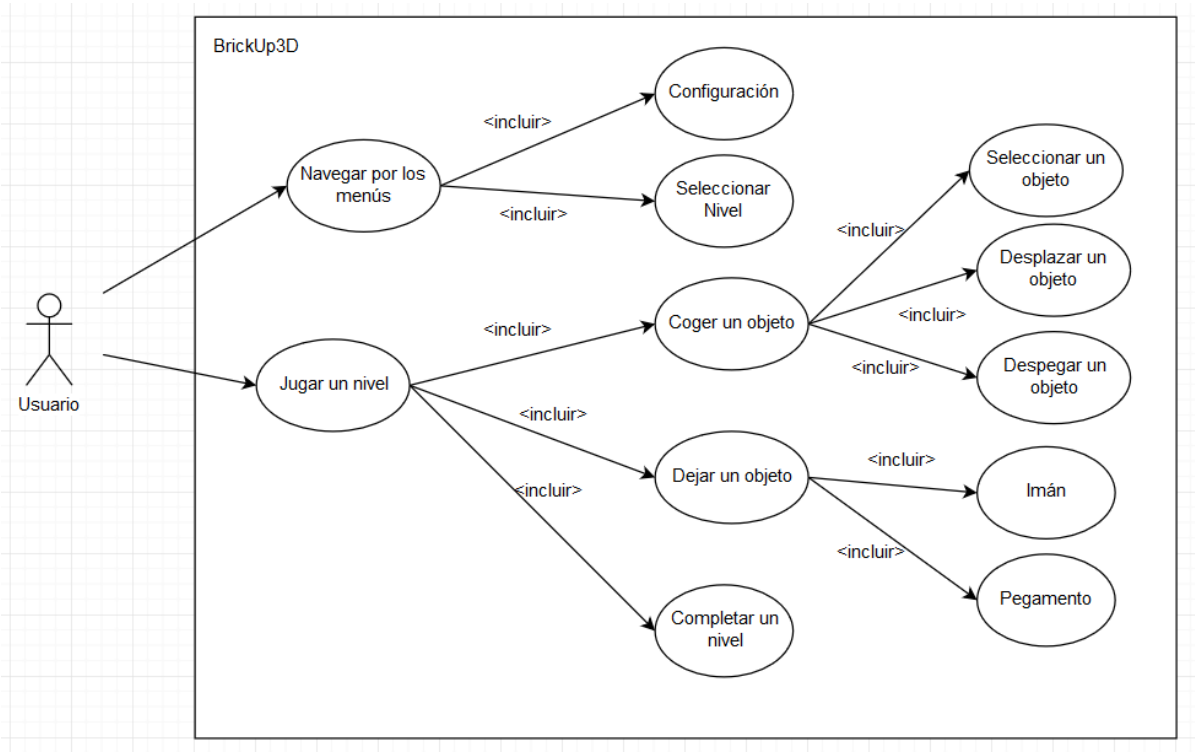

Ilustración 1: Diagrama de casos de uso

Nota: los casos de uso de imán y pegamento se explicarán más adelante.

2.3.- Diagrama de clases:

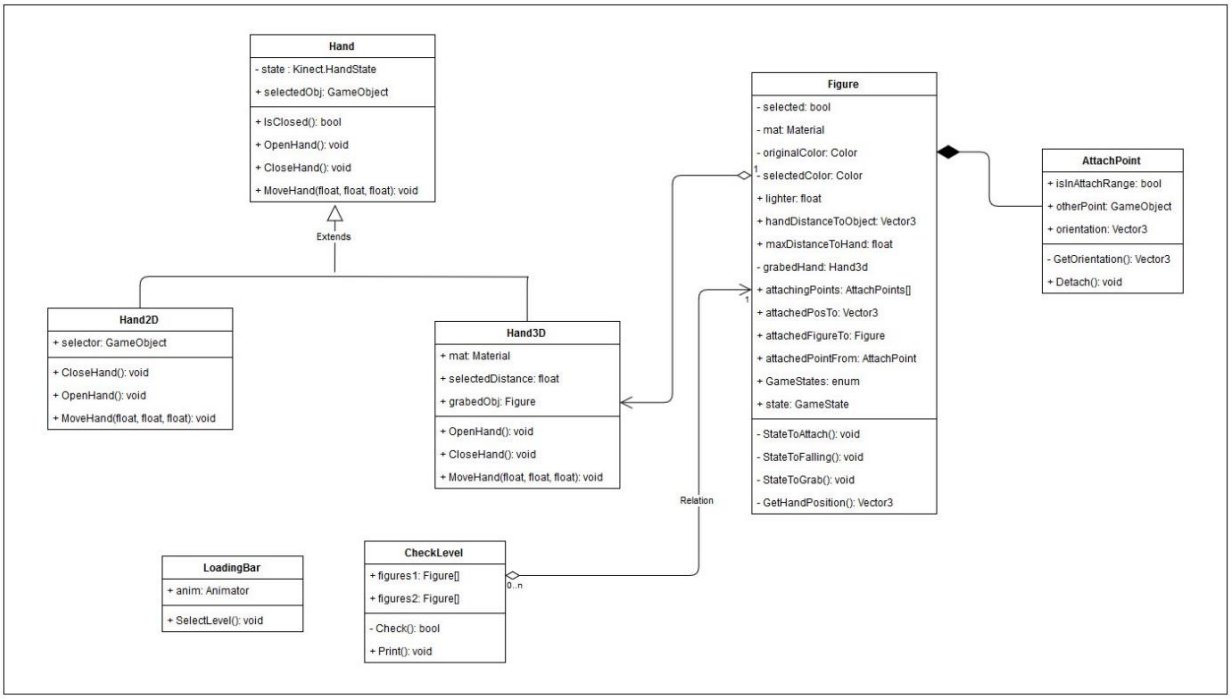

Ilustración 2: Diagrama de clases

Uno de los objetos principales son las manos. Hay 2 tipos de manos, una para el menú de selección de nivel, que está en 2 dimensiones (Hand2D), y otra para jugar moviendo las figuras en 3d (Hand3D). Ambas extienden de Hand, de esta manera se puede emplear el mismo script para que la Kinect les cambie la posición y estado a las manos independientemente de si son en 2d o 3d. Hand también tiene "selectedObject" que en el caso de una "Hand2D" hará referencia a un nivel y en el de las Hand3D a una figura. Por último, Hand proporciona unos métodos comunes a ambos tipos de manos, pero con funciones algo distintas en cada una.

La mano en 2d (ver: Ilustración 3: Mano 2D) además tiene un selector que se encarga de indicar al usuario el nivel que se está seleccionando y que es controlado por la mano. La mano en 2d recibe la posición en 3d de la Kinect, pero redefine el método MoveHand() que normaliza el desplazamiento de la mano por el menú en 2 dimensiones para que sea cómodo y sin gestos bruscos. También ignora la posición del eje Z, puesto que el movimiento es bidimensional.

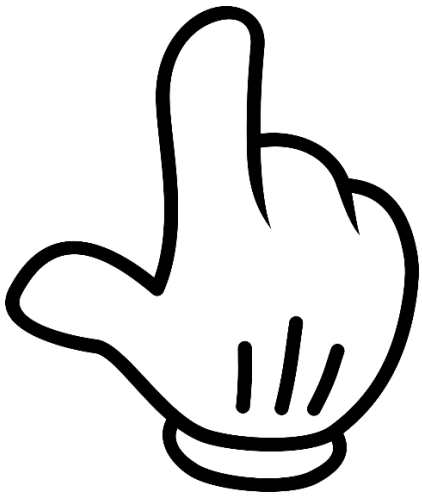

Ilustración 3: Mano 2D

La mano en 3d (ver: Ilustración 4: Mano en 3D) tiene su material y añade al "selectedObject" una "selectedDistance" para poder cambiar el objeto seleccionado al más cercano. Por último, tiene un "grabedObj" que hace referencia a la figura que tiene agarrada, null si no tiene ninguna.

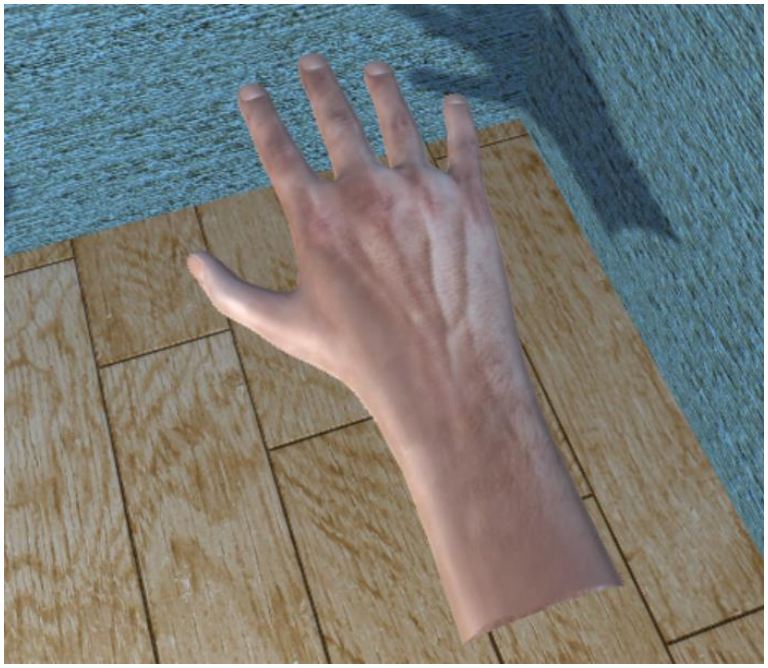

Ilustración 4: Mano en 3D

Otro objeto muy importante es “Figure”. Este objeto representa las figuras de la escena, tiene los siguientes atributos:

- Un booleano que le indica si está seleccionada por alguna mano
- Su material
- su color original
- su color cuando está seleccionada (más claro que el color original)
- el valor que indica la diferencia entre su color original y el color de seleccionada
- la distancia a la que tiene que ir de la mano cuando es agarrada
- la distancia máxima a la que tiene que separarse la mano para despegarse de otra figura (común a todas las figuras)
- la mano que le está agarrando (null si no es ninguna)
- los puntos de acople que pertenecen a esta figura
- la posición en la que debe de estar cuando se atrae por el imán a otro objeto
- la figura a la que se pega (null si no es ninguna)
- el punto de acople por el que está pegada a su padre
- los estados de la figura (ver Ilustración 9: Diagrama de transición de estados del objeto Figure)
- el estado actual de la figura de entre los estados posibles

También contiene métodos para cambiar los parámetros entre estados y calcular la posición en la que debería colocarse cuando es cogido por la mano.

Por último, están los “AttachPoints” que forman parte de la figura y representan los puntos de acople para la funcionalidad del imán y pegamento de las que se hablará posteriormente. Estos objetos tienen un booleano que representa si tienen otro punto de acople en rango para acoplar, el punto que está en rango (null si no hay ninguno en rango) y la orientación para acoplarse con otros puntos compatibles. También incluyen un método para calcular su orientación y otro para borrar las referencias a otros puntos de acople y sus distancias.

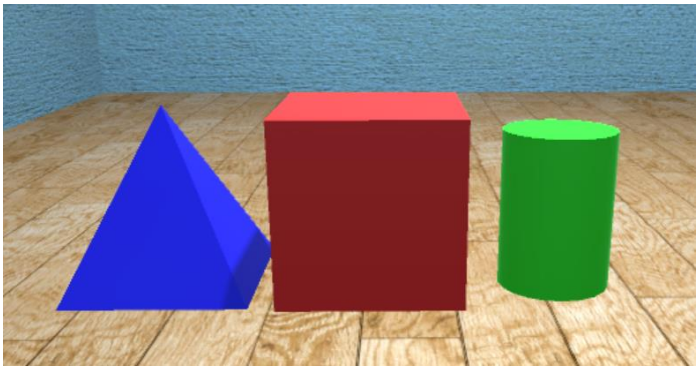

Ilustración 5: Figuras de ejemplo

3.- Funcionalidades:

3.1.- Configuración:

En el menú de inicio se tiene un botón en el que pone “Options” que al clicar con el ratón lleva al usuario a una ventana donde puede realizar alguna configuración de la aplicación. Las opciones son:

- “Dominant Hand”:

  Esta opción le permitirá al usuario indicar cuál es la mano con la que mejor se maneja. Durante el juego podrá utilizar ambas así que no cambiará su experiencia construyendo. Esta opción solo se aplica a la mano del menú de selección de nivel que, dependiendo de qué mano haya escogido, navegará con una o con otra. Básicamente se mostrará solo un objeto al que se le asignará la posición del *Joint* de la mano que el usuario haya escogido.

- “Hand color”:

  Con esta opción el usuario podrá cambiar el color de la mano que controla durante el juego. Esta opción sirve solamente para mejorar la inmersión del jugador mientras construye con *BrickUp3D*. La mano del menú de selección de nivel es un guante y no cambiará. Para hacer esto cambiaremos el material del modelo de la mano a uno más oscuro en caso de que elija la opción de más oscura.

3.2.- Seleccionar un nivel:

Para acceder a la selección de niveles simplemente se deberá clicar en el botón de “Play” del menú inicial (Ilustración 6: Menú inicial).

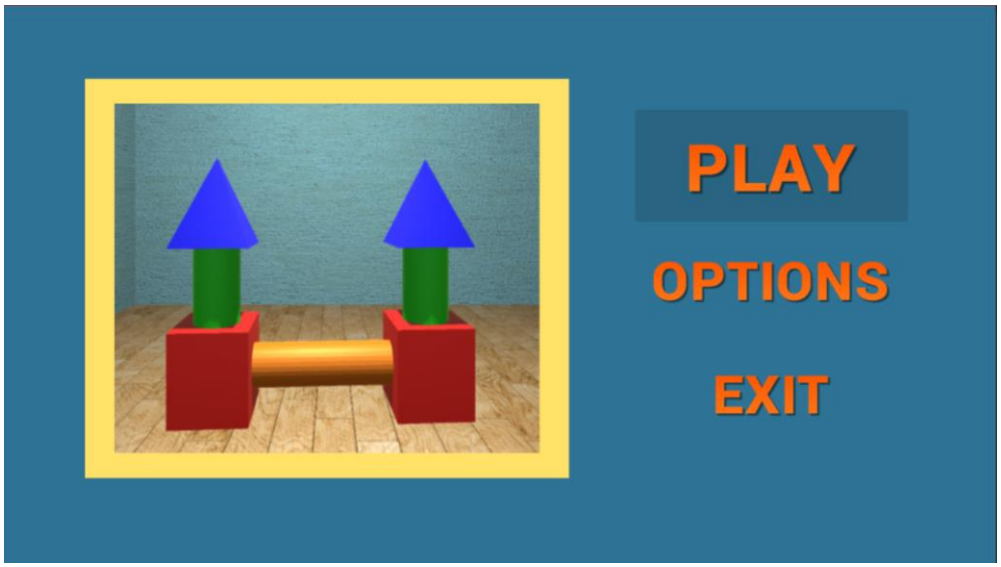

Ilustración 6: Menú inicial

En menú de selección (Ilustración 8: Selección de nivel) se muestran 6 rectángulos en los que pone “Level” seguido del número de nivel que le corresponde a cada uno. El

número es un identificador para diferenciarlos, pero además por lo general un número mayor supondrá una mayor dificultad.

Para navegar se tiene un objeto en 2D (Ilustración 3: Mano 2D) controlado por los gestos que lee la Kinect. Se mueve parecido a la mano que se controla durante la pantalla principal del juego, pero en 2 dimensiones. Para moverla simplemente se le asigna al GameObject la posición normalizada del *Joint* de la mano seleccionada en la configuración como "Prefered Hand". Se normaliza para que sea cómodo moverse por el menú con gestos que no requieran mucho esfuerzo ni movimiento y el eje Z no se asigna ya que el cursor del menú se va a desplazar únicamente en los ejes X e Y.

Para seleccionar un nivel simplemente se pone la mano encima del rectángulo que representa ese nivel y se cierra hasta que se llene una barra de carga (Ilustración 7: Barra de carga completándose). Esta barra de carga es para evitar seleccionar niveles sin querer, puesto que a veces la Kinect se equivoca y capta mal el estado de la mano.

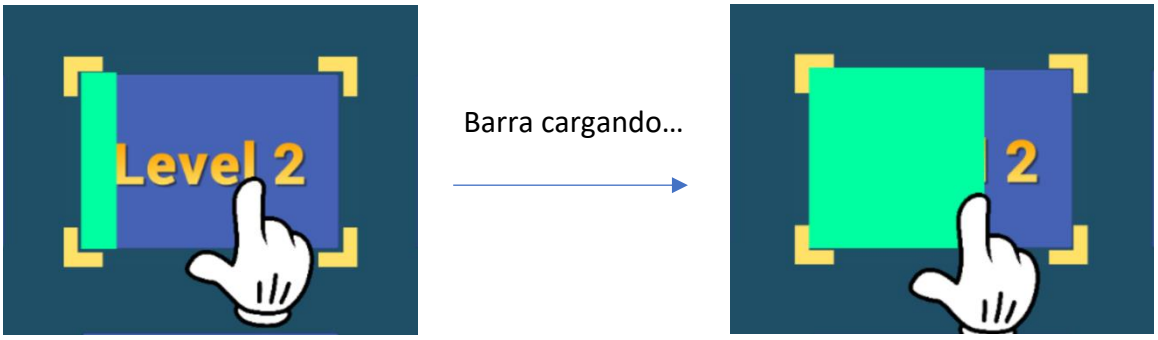

Ilustración 7: Barra de carga completándose

Para implementar esto se emplea un RayCast desde la posición del objeto que representa la mano (realmente un poco más arriba para que se seleccione con la punta del dedo) hacia el frente y si detecta un botón entonces la mano guarda el botón seleccionado. Para que el usuario sepa también que ese nivel está seleccionado hay un objeto animado que recubre las esquinas del botón que esté seleccionado.

Ahora cuando se detecta que el estado del *Joint* de la mano está en *Kinect.HandState.Closed* se coge el *AnimatorController* de la barra de carga y se le asigna a True una variable que le indica que debe estar ejecutando la animación de carga. Si el *RayCast* de la mano deja de chocar con ese botón antes de que la animación haya finalizado se le asigna a la variable del *AnimatorController* de la barra de carga False y la animación se detiene, por lo que la barra de carga vuelve a estar a 0. Por otro lado, si se deja que se ejecuta la animación entera, la barra llegará hasta el 100% cubriendo todo el recuadro y un *AnimationEvent* al final de la animación llamará a un método que cambia la escena a la del nivel al que hace referencia ese botón.

Ilustración 8: Selección de nivel

3.3.- Manipular una figura:

Para comprender el comportamiento de las figuras (Ilustración 5: Figuras de ejemplo) en cada una de las funcionalidades y el significado de cada uno de sus estados se adjunta el Diagrama de transición de estados. La transición entre los estados se explica en las distintas funcionalidades:

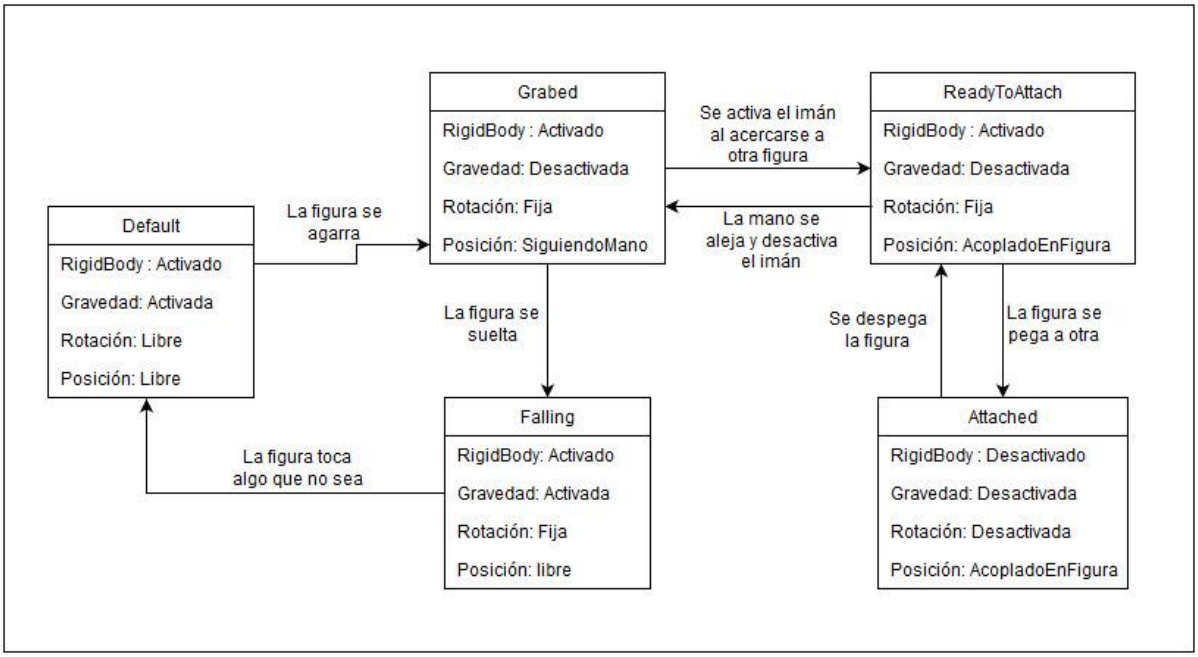

Ilustración 9: Diagrama de transición de estados del objeto Figure

A continuación, se presentarán las funcionalidades que se implementaron.

3.3.1- Coger un objeto:

3.3.1.1.- Versión inicial:

Para poder montar estructuras es necesario poder coger las figuras. Inicialmente la manera en la que se cogían los objetos era cuando se detectaba que la mano estaba

cerrada y su *Collider* colisionaba con el *Collider* de la figura, entonces la figura pasaba del estado “Default” al estado “Grabed”. Durante el estado “Default” la figura no realiza ninguna acción, pero cuando la figura está en estado “Grabed” se le asignaba la posición de la mano (un poco más abajo para que diera el efecto de que es agarrado por ella).

Esta manera de hacerlo tenía sus problemas, primero si se le asignaba a la figura directamente la posición de la mano había veces que se creaban choques entre elementos muy fuertes puesto que de repente aparecía una figura que se superponía a otros elementos y hacía que estos volaran por los aires.

3.3.1.2.- Resultado final:

La manera de solucionar esto fue en vez de asignarle a la figura la posición directamente, utilizar la función *Lerp* de *Vector3* que permite desplazar un objeto poco a poco en vez de teletransportarlo asignándole la posición.

El segundo problema era que al intentar acercar la mano para coger un objeto era necesario colisionar con él y empujarlo, lo que podía suponer todo el derrumbamiento de la construcción. Se arregló implementando una ayuda al usuario que consistía en seleccionar un objeto que estuviera a cierta distancia (ver Ilustración 10: Figura antes y después de ser seleccionada) y cogiéndolo si se cerraba la mano.

Para llevar esta ayuda a cabo inicialmente se pensó que se podía calcular las distancias a cada figura desde la mano, pero habría que hacer esta comprobación en cada iteración para cada figura.

Finalmente se decidió que era más sencillo asignarle otro *Collider* a la mano, pero con la opción *isTriggered* en True y aprovechar los callbacks de *OnTriggerStay* y *OnTriggerExtit*. Así se conseguía que los objetos detectaran la mano sin que fueran empujados, únicamente cuando la figura entrara en este *Collider* de la mano. Ahora cuando una figura detectara que entraba en contacto con este nuevo *Collider* en el

método *OnTriggerStay*, si era la figura que se encontraba más cerca de la mano se le cambiaba su color por uno más claro para que el usuario supiera que esa figura estaba seleccionada. Para saber quién es la figura más cercana a la mano esta guarda la figura que tiene seleccionada y la distancia a la que se encuentra. Por otro lado, si la figura detecta que el *Collider* de la mano sale en *OnTriggerExit* se deselecciona si era la figura seleccionada y se borran las referencias de esta figura en la mano.

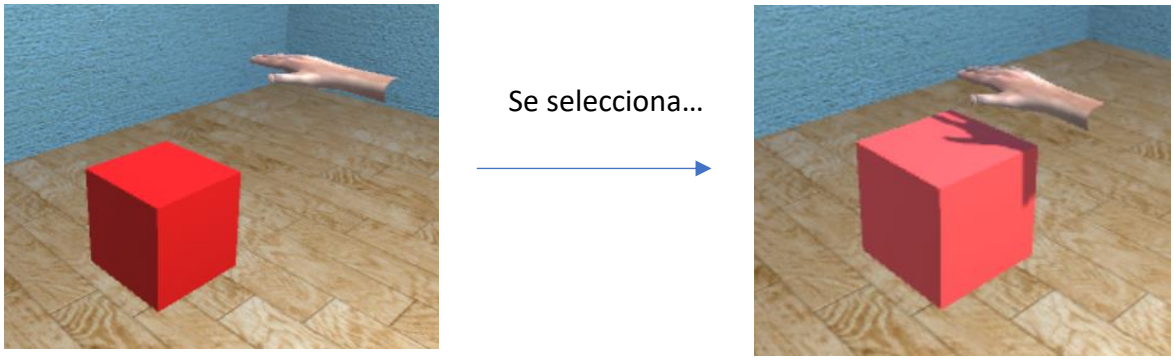

Ilustración 10: Figura antes y después de ser seleccionada

Se utiliza *OnTriggerStay* en vez de *OnTriggerEnter* porque podría resultar que el *Collider* de la mano colisionara con 2 figuras y en un momento se encontrara más cerca de una y en otro de otra sin dejar de colisionar con ambas y también debería cambiarse la figura seleccionada. Una mano solo puede seleccionar una figura al mismo tiempo.

Otras medidas que se llevaron a cabo hasta llegar a la versión final de esta funcionalidad fueron:

- fijar la rotación de la figura cuando fuera agarrada para que no girara mientras era llevada
- Asignar a la figura que se cogiera su rotación inicial para colocarlas rectas.

Se hablará de la funcionalidad de “Despegar un objeto” en el siguiente punto cuando se explique que es “Pegar un objeto”.

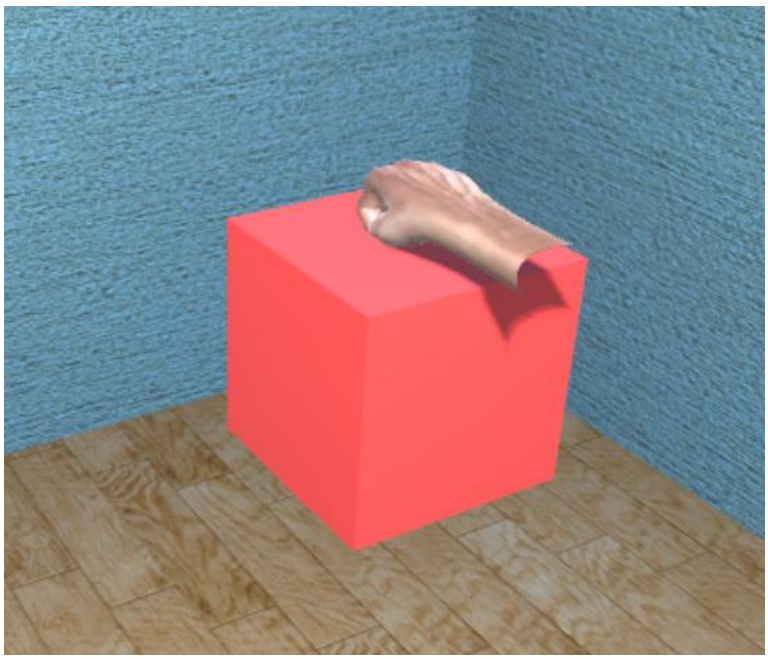

Ilustración 11: Figura siendo cogida por una mano

3.3.1.3.- Limitaciones:

En un inicio también se quiso poder rotar las figuras que estaban siendo sujetadas por una mano. Por desgracia esta medida no se llevó a cabo ya que se consideró que la precisión de la Kinect al detectar la orientación de las manos no era lo suficientemente precisa para que fuera cómodo.

3.3.2.- Dejar una figura:

3.3.2.1.- Versión inicial:

Al igual que coger los objetos, es necesario poder colocarlos para poder completar un nivel.

Una vez se ha cogido un objeto, como se ha descrito en la funcionalidad anterior, para colocarlo simplemente debe abrirse la mano, esto ocurre cuando el *Joint* de la mano que la sujetaba se encontraba en estado *Kinect.HandStates.Open*.

En la versión inicial simplemente se cambiaba el estado de una figura a "Default" que reactiva la gravedad para el objeto y hace que deje de seguir la mano. Con esto supuestamente bastaría, pero se daban una serie de sucesos que hacían excesivamente complicado jugar y que se requiriera una precisión demasiado elevada, sobre todo al apilar las figuras ya que, por ejemplo, si la Kinect detecta mal la posición de la mano en un instante puede empujar una pieza y llegar a destruir todo lo que se había construido.

3.3.2.2.- Versión final:

Como se hizo con la funcionalidad de coger objetos, se implementaron ayudas al usuario que facilitaran esta tarea.

Uno de los problemas que surgieron fue que al soltar una figura esta muchas veces giraba antes de caer y dificultaba mucho la tarea de apilar. Se arregló añadiendo un nuevo estado para la figura: "Falling" este estado indicaba que una figura estaba cayendo, es similar a "Default" solo que mantiene la rotación fija hasta que colisiona con cualquier cosa que no sea una mano. En este momento su estado pasa a "Default".

Otro problema es que, aunque sea un juego dirigido para niños, formar construcciones con la Kinect requiere bastante precisión y puede ser algo complicado.

Por este motivo se implementó la ayuda del imán. Esta ayuda sirve para poder apilar los objetos rectos y centrados (ver: Ilustración 12: Figuras apiladas centradas gracias al imán). Cuando una figura está cogida por una mano y se acerca a una cierta distancia a otra a la que se podría apilar se activa el imán, la figura cogida por la mano se posiciona directamente acoplada en el centro a la figura a la que se había acercado y si la mano se abre ya se queda posicionada de esa manera. Por otro lado, si la mano permanece cerrada y se aleja lo suficiente, la figura se desacoplará y volverá a seguir a la mano.

Para implementar esta ayuda se utilizaron puntos de acople o "AttachPoints". Estos puntos son GameObjects que posee cada figura e indican los lugares donde una figura puede acoplarse. Estos puntos poseen un *Collider* que detecta cuando el punto de acople de otra figura, que tiene una rotación compatible, entra en su rango. Cuando esto sucede el punto de acople almacena el punto de la otra figura y su distancia, para tener acoplado el punto más cercano. Cuando una figura está en estado "Grabed" y detecta que uno de sus puntos de acople ha captado a otro, inmediatamente se le suma a su posición la diferencia entre las posiciones de los dos puntos para desplazarse en el lugar óptimo para estar acoplado a esa figura y su estado pasa a "ReadyToAttach".

Una vez que una figura está en el estado "ReadyToAttach" si la mano se abre pasa al estado "Default". Si por el contrario la mano permanece cerrada y se aleja más allá de una distancia prefijada ("MaxDistanceToHand") el objeto volverá al estado de Grabed y borrará de sus puntos de acople los puntos de la figura a la que estaba acoplada.

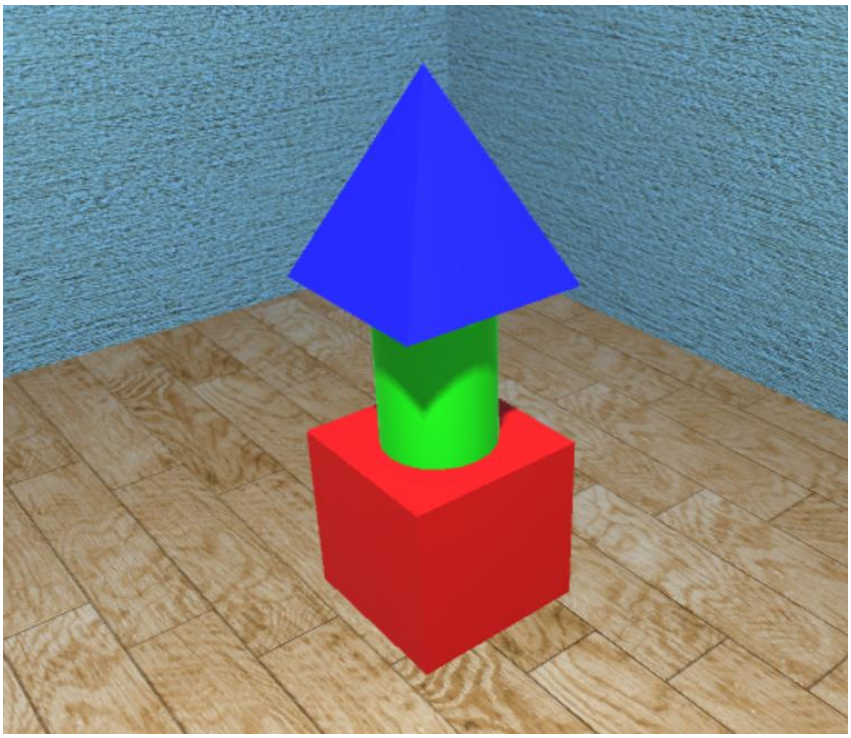

Ilustración 12: Figuras apiladas centradas gracias al imán

La implementación de la ayuda del imán dio lugar a la idea de la ayuda del "pegamento". La ayuda consiste en que los objetos se unan y se conviertan en uno solo cuando se utiliza el imán para unirlos (ver: Ilustración 13: Figura posible por el uso del pegamento). Esta ayuda permitiría poder construir una construcción más grande por partes y unirlas al final para construir la definitiva.

Para hacer esto en vez de pasar al estado "Default", cuando una figura está en el estado "ReadyToAttach", pasar a otro estado nuevo llamado "Attached" que indicaría que la figura está pegada. Cuando el usuario coge una figura en estado "Attached" inmediatamente pasa a "Grabed" y de esta manera se puede despegar. Para mover todo el conjunto de figuras debe de cogerse el padre que es la pieza a la que se ha pegado.

Cuando la figura pasa a estar en el estado de "Attach" se le asigna de padre la figura a la que se ha pegado. Además, inicialmente se le puso su *RigidBody* en *Kinematic* lo cual impedía que fuera afectada por las físicas al chocar con otros objetos

y de esta manera permanecer inmóvil con respecto a su padre. Sin embargo, esto ocasionaba que la mano atravesara las figuras que se habían acoplado lo cual resultaba incoherente con un entorno basado en la física por lo que más adelante hubo que solucionarlo.

En la versión final en vez de poner los *RigidBody* en *Kinematic*, se les borró este componente a las figuras en el momento en el que quedaban acopladas y posteriormente se les volvía a crear cuando fueran a desacoplarse, solucionando así que se traspasaran con la mano.

El problema ahora era que cuando se acercaba la mano a la figura hija la figura que detectaba a la mano era únicamente la figura padre y por lo tanto no era posible desacoplarlas. Para poder recuperar la funcionabilidad de desacoplar las figuras se comprobó en el método *OnTriggerEnter* de la figura padre, cuando la mano estaba cerrada, si en efecto era el padre de más figuras y en ese el caso asignarles a todos sus hijos un *RigidBody* y salir del método. De esta manera ahora si serían los hijos los que detectaran a la mano y si se cogía una figura se podía desacoplar. Después de coger la figura esta se encarga de quitarles el *RigidBody* a todos los hijos de su anterior padre para dejarlo como estaba.

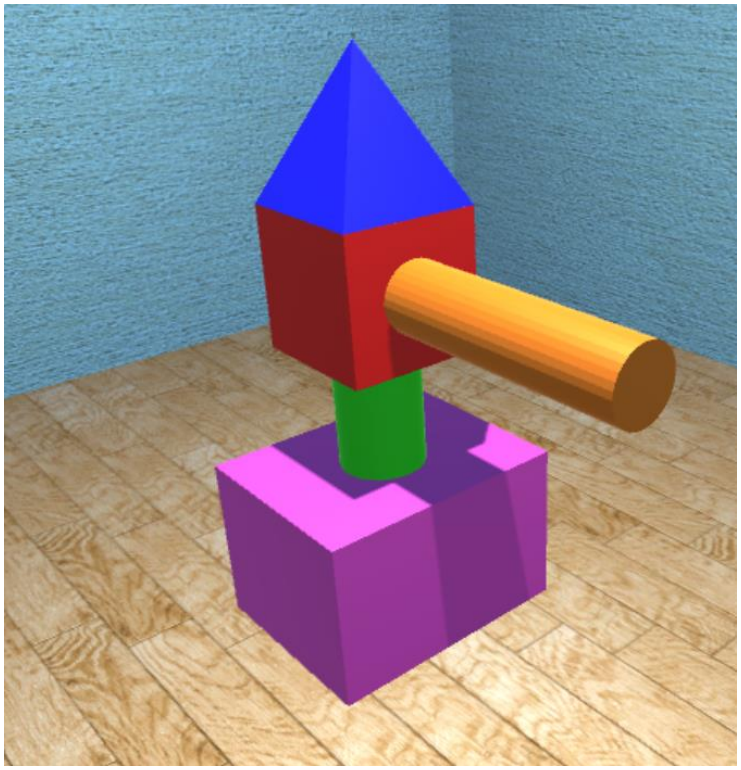

Ilustración 13: Figura posible por el uso del pegamento

3.3.2.3.- Limitaciones:

Cuando se implementaron las ayudas se pensó en añadir modos de dificultad de manera que cuanta más dificultad se escogiera menos ayudas se tendrían activadas, pero la mecánica del pegamento resultó interesante porque abría el abanico de estructuras que podían crearse y que sin la ayuda del imán era imposible llevar a cabo. Por este motivo se dejaron las ayudas de manera permanente y no se implementaron niveles de dificultad directamente. Sin embargo, conforme se aumenta de niveles también aumenta la dificultad, y no solo por la complejidad de la estructura a construir si no que por ejemplo en algún nivel se invierte el movimiento de la mano en algún eje, lo que requiere un extra de coordinación y esfuerzo mental para construir.

3.5.- Completar un nivel:

La última de las funcionalidades que ofrece el juego es la capacidad de completar los distintos niveles. Un nivel se completa cuando se ha construido con éxito una estructura que sea idéntica a la que se muestra en la pared del fondo.

Para comprobar si el nivel se ha completado se emplea un objeto “CheckLevel”. Este objeto se aprovecha del mecanismo de los puntos de acople para averiguar si el nivel se ha completado.

Cuando el nivel está completado con éxito aparece un texto, en la pared del fondo donde está la imagen de la solución, en el que pone “SUCCESS”.

3.5.1.- Idea inicial:

Inicialmente su funcionamiento consistía en comparar 2 *Arrays* del mismo tamaño que representan las parejas de figuras que tienen que estar unidas. Por ejemplo, si están unidos un cubo y un cilindro en la figura final entonces habrá que agregar en un array el cubo y en el otro en la misma posición el cilindro. En el momento de comprobar si se llega a la solución correcta se recorrerán ambos *Arrays* al mismo tiempo comprobando que los puntos de acople de las figuras del primer *Array* están unidos con una figura con el mismo nombre que la figura pareja (con la misma posición en el segundo *Array*).

3.5.2.- Limitaciones de idea inicial:

Esta manera de comprobar un nivel puede dar algunos problemas. Hay que tener en cuenta que una figura puede pegarse a otra en una cara u otra y aún y todo

podría estar bien. También hay que tener en cuenta que hay figuras que pueden ser intercambiables, por ejemplo, si hay 2 torres formadas por 2 cilindros y 2 pirámides puede ser que un cilindro esté acoplado a una pirámide o a la otra y en ambos casos la figura final sería correcta. Para poder permitir este tipo de situaciones y que el comprobador de nivel funcione correctamente deben seguirse las siguientes condiciones de manera estricta.

Si las dos figuras que forman la pareja son únicas no importa en que array se coloque cada una.

Para permitir que haya 2 figuras iguales que puedan ser intercambiables (por ejemplo 2 cilindros exactamente iguales) para acoplarse a una figura única (que solo existe una de ese tipo y tiene un nombre único), estas deben tener el mismo nombre y colocarse en el primer array para asegurarse de que ambas estén acopladas a la figura única.

Si por el contrario hay 2 figuras repetidas que se acoplan a otras 2 repetidas no importa en que array se coloque cada uno siempre y cuando las figuras iguales estén en el mismo array.

Esta forma de rellenar el CheckLevel y de trabajar resultó demasiado difícil y además requería tener mucho cuidado y cumplir unas condiciones muy complicadas, por ello se cambió el método para comprobar si el nivel estaba resuelto.

3.5.3.- Idea final:

Para el resultado final se cambió la representación de datos. En vez de utilizar 2 *Arrays* se utiliza un *ArrayList* que representa una cadena de figuras unidas entre sí. Por ejemplo, si deberían estar unidos un cubo y un cilindro, el primer elemento será un *String* “cube” y el segundo “cylinder” o viceversa. Por otro lado, si una figura está unida a varias directamente, en vez de poner el nombre de la figura, se pondrá otro *ArrayList* con el conjunto de figuras al que está unida.

3.5.4.- Limitaciones idea final:

Como Unity no permite rellenar un *ArrayList* en el inspector, para indicarle al "CheckLevel" de cómo es la figura final utilizamos un *String* con un cierto formato. Primero elegimos una figura cualquiera por la que empezar, después para separar las figuras unidas se usa un guion y si no tiene unida una única figura se utiliza paréntesis y se separan las figuras o cadenas con comas.

Un ejemplo, para la figura de la Ilustración 13: Figura posible por el uso del pegamento, sería: "prismaRectangular-cilindroVerde-cubo-(pirámide, cilindroNaranja)"

4.- Resultados, análisis y discusión:

4.1.- Restricciones tecnológicas:

La principal restricción tecnológica que se ha dado durante el desarrollo del proyecto ha tenido relación con la Kinect. Si se conoce su funcionamiento y manera de actuar puede jugarse a este juego con bastante precisión y comodidad, pero para un usuario que lo desconoce el manejo de las manos puede resultar una tarea difícil.

El motivo de que resulte difícil es que dependiendo de la posición de las manos la Kinect no las detecta correctamente y no puede saber si están cerradas o abiertas. Para que las detecte bien tiene que ver la palma de la mano, lo cual deja 2 soluciones posibles:

- En vez de apuntar las palmas de las manos hacia el suelo, apuntarlas hacia la cámara.

El problema de esta solución es que en la realidad las manos se colocan con las palmas hacia abajo para coger figuras del suelo por lo que pierde algo de realismo. Por otro lado, el juego responde mucho mejor empleando esta medida.

- Poner la cámara en el suelo y apuntar hacia arriba.

  Con esta solución la cámara ya apuntaría a las manos, aunque estas estuvieran con las palmas hacia abajo. El problema que puede dar esta solución es que limita ligeramente el espacio en el que puedes desplazarte para controlar las manos virtuales.

Otra restricción que ofrece utilizar una cámara para gestionar el movimiento es la de captar la rotación de las manos. La Kinect se fija en los dedos y en los brazos para averiguar si una mano está girada hacia un lado u otro. Es cierto que alguna de las rotaciones se capta con algo de precisión cuando la mano del usuario está abierta, pero cuando la mano está cerrada, que es el momento donde más importa para obtener su rotación, la precisión empeora enormemente porque no puede reconocer la posición de los dedos fácilmente.

Por último, también da un poco de problemas tener las manos muy cerca entre sí, en el juego se nota que se dan algunos movimientos extraños cuando esto pasa.

4.2.- Contexto tecnológico:

Actualmente la tecnología está en auge y cada vez se emplea más para ámbitos de salud y bienestar. Una manera de aplicar la tecnología a la salud, sobre todo para los niños, es por medio de juegos entretenidos que les ayude a realizar ejercicios y que permiten evaluar sus capacidades. *BrickUp3D* formaría parte de este tipo de juegos que cada vez cogen más importancia en la vida de las personas.

En este momento además está cogiendo mucha fuerza la realidad virtual, no solamente para los juegos, sino para fines terapéuticos y quizás más adelante se

crearán varios proyectos en los cuales se usará junto con el cuerpo como controlador con una cámara como la Kinect.

4.3.- Cumplimiento de los objetivos y resultado final:

Considero que los objetivos principales se han cumplido con éxito. Al comienzo de la implementación de las funcionalidades daba la impresión de que la aplicación podría quedar un poco inestable ya que el control mediante el cuerpo por medio de una cámara no es tan exacto a como estamos acostumbrados con los controladores habituales, sin embargo finalmente mediante la implementación de las ayudas, algunos retoques y teniendo en cuenta algunas consideraciones como la de apuntar la palma de la mano a la Kinect se ha obtenido una versión final muy estable y con una jugabilidad muy buena.

También ha quedado un juego atractivo para ser jugado por niños, lo cual es muy importante para que lo usen y poder comprobar así su percepción de las físicas.

5.- Bibliografía y referencias:

- Microsoft, información de la Kinect:
  https://www.microsoft.com

- API Kinect:
  https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn799271(v=ieb.10)

- Definición de algunas herramientas:
  https://es.wikipedia.org

- Unity tutoriales:
  https://unity3d.com/es

- Unity Scripting API:
  https://docs.unity3d.com/ScriptReference

- Unity preguntas y respuestas:
  https://answers.unity.com/index.html