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

Aplicación de Realidad Virtual en Unity para Android con fines educativos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Rubén Baztán Larrea

Oscar Ardaiz Villanueva

Pamplona, 5 de septiembre de 2018

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Índice

1. Introducción y presentación del proyecto

En esta memoria se describe la realización del Trabajo de Fin de Grado realizado para culminar la carrera de Ingeniería Informática. El proyecto se ha elaborado con la ayuda del profesor Oscar Ardaiz Villanueva, que ha sido de gran ayuda en todo momento.

Para empezar, quiero explicar en qué consiste el proyecto, cómo llegué a la idea que lo originó y, en definitiva, cuales fueron los motivos para escogerlo. Desde hace bastante tiempo, cuando empecé a definir en mi mente cómo deseaba que fuese el trabajo más importante de mi etapa como estudiante, tenía la idea de que quería que se tratara de algo relacionado con la educación.

El motivo es que yo, a pesar de estudiar ingeniería, estoy muy ligado a ese mundo, ya que al mismo tiempo que empecé la carrera, también comencé a ser monitor de tiempo libre y ahora mismo es una de las cosas que más me llenan en la vida. Así que me pareció bueno pensar en el disfrute que me supondría su realización, mezclando dos de los temas que más me gustan. También quería que fuese algo original y diferente.

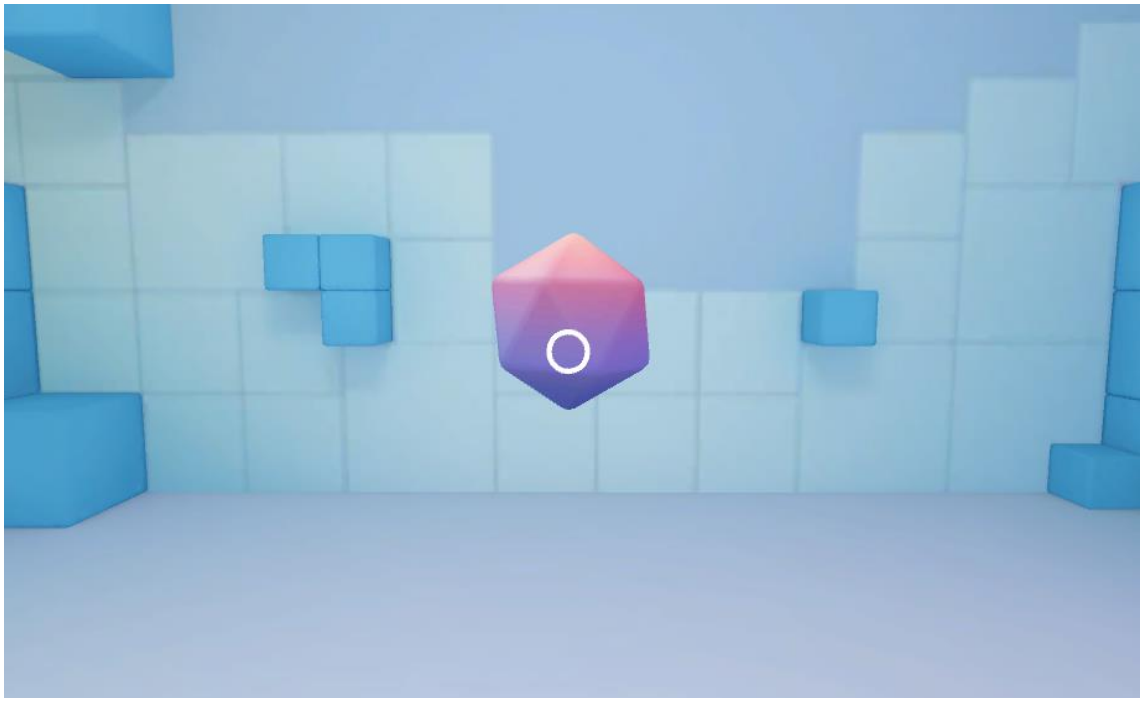
En febrero de 2018, mientras exploraba las propuestas para TFG disponibles en la web de la universidad, me percaté de la presencia de un proyecto que consistía en la elaboración de una aplicación para móvil que consistía en un juego con el cual, los estudiantes pudieran asimilar conceptos de biología (era una propuesta de dos profesores, uno de informática y otro de biología). Hablé con Oscar, que era uno de los profesores que proponía este proyecto, pero me dijo que ese trabajo era de otro año y que ya se había realizado. No obstante, también me comentó que me pasara por su despacho para ver si se nos ocurría otro proyecto parecido. Me gustaba el estilo del proyecto antes descrito, así que accedí.

Cuando me reuní con Oscar, hablamos sobre cómo nos gustaría que fuese el proyecto. Yo expuse las ideas que he mencionado anteriormente y él, pensando en lo que le había dicho a cerca de la originalidad del proyecto, me sugirió que podíamos probar a utilizar la tecnología de Realidad Virtual.

La Realidad Virtual es algo nuevo, original, que despertaba mi curiosidad y desde un primer momento, me pareció una ocurrencia interesante. Oscar me contó que Google había desarrollado librerías para la programación de aplicaciones VR para Android e IOS y que estas se podían aplicar en Unity. El tema de usar Unity es algo que había tenido en mente desde un primer momento. Por aquel entonces estaba cursando la asignatura de Sistemas Multimedia y acabábamos de empezar a cacharrear con este motor de videojuegos, lo cual me gustó mucho. Además, me pareció buena idea el hecho de programar una aplicación para móvil, ya que actualmente es el dispositivo más utilizado con diferencia.

Así que llegamos a la conclusión de que el trabajo consistiría en una aplicación VR, en principio pensada para Android y que tuviera algún tipo de contenido educativo. Y así fue como empezó todo.

Después de la reunión con Oscar estuve unos días probando las librerías de Google VR y probando algunas de las aplicaciones que traen consigo como ejemplo. Una de ellas consistía en un juego en primera persona, en el cual te encuentras en una habitación extraña, donde tienes que encontrar diferentes figuras. Para encontrarlas lo único que tienes hacer es girar para ver la habitación por completo o hasta dar con la figura y tocar la pantalla (en el simulador de Unity simplemente tenías que hacer clic).



Al cabo de unos días, tras estar un tiempo pensando, llegué a la idea de cuál podía ser la forma de la aplicación, la lógica. Se trata de un juego que está basado en un Escape Room, el ahora de moda pasatiempo en el cual el objetivo es escapar de una habitación en un tiempo determinado, a base de resolver acertijos y encontrar llaves para abrir puertas. También me he basado en películas como "*Cube*" o "*La habitación de Fermat*", en la que los personajes se encuentran en una situación similar a la que encontramos en un *escape room*, solo que el peligro de no resolver los enigmas es real.

2. Explicación del funcionamiento del programa

Cuando iniciamos el juego, nos encontramos en una habitación oscura y solo contamos con una linterna para iluminar malamente la instancia allí donde miramos. En la habitación, a priori, no hay nada, excepto tres puertas. Solo una de ellas te conducirá a la siguiente sala y te acercará a la libertad, las otras dos te conducen a un pozo sin fondo y te hacen perder el juego.

Cada puerta tiene una inscripción; las inscripciones sirven para saber cuál es la puerta correcta, pero con esto no basta. En algún lugar de la habitación hay otra inscripción, o bien un dibujo que es una clave para resolver el enigma. Solo hay que relacionar esa pista con las inscripciones de las puertas para saber cuál es el camino a seguir. La pista también puede ser un sonido y en tal caso no es necesario buscarla por la habitación, sino que lo escucharemos en todo momento. Se trata de avanzar a base de resolver acertijos de una habitación a otra, hasta que llegamos al final y conseguimos la ansiada libertad.

La dificultad radica en buscar e interpretar las pistas que se encuentran en un lugar aleatorio de la habitación, pero además tendremos un límite de dos minutos para resolver el enigma. De lo contrario, la habitación desaparecerá y caeremos al vacío de la misma forma que si hubiésemos escogido la puerta incorrecta. Justo encima de las puertas se halla un marcador electrónico que nos informa del tiempo que nos queda en todo momento.

El juego en sí es sencillo. El tema es el trasfondo de los enigmas que se plantean. No son enigmas extraños ni difíciles de resolver, si no que se trata de una especie de "trivial" a cerca de conceptos que se aprenden en los colegios hoy en día. Por tanto, con esto conseguimos una aplicación que permite a los niños y niñas asimilar conceptos aprendidos en clase de una forma entretenida, moderna y misteriosa (algo que les atrae).

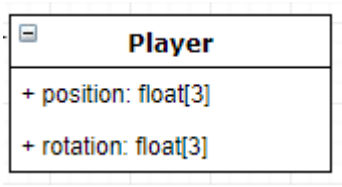
Sin embargo, aunque actualmente los acertijos planteados tienen esta índole, se podría extrapolar otro tipo de enigmas que puedan a atraer al público adulto y no lo limiten solo al campo de la educación.

Se nos ocurrió también plantear un modo invertido, en el cual la cámara girara en el sentido contrario al que se mueve nuestra cabeza. Esto nos permite ejercitar no solo nuestras habilidades intelectuales, si no también nuestras habilidades motrices.

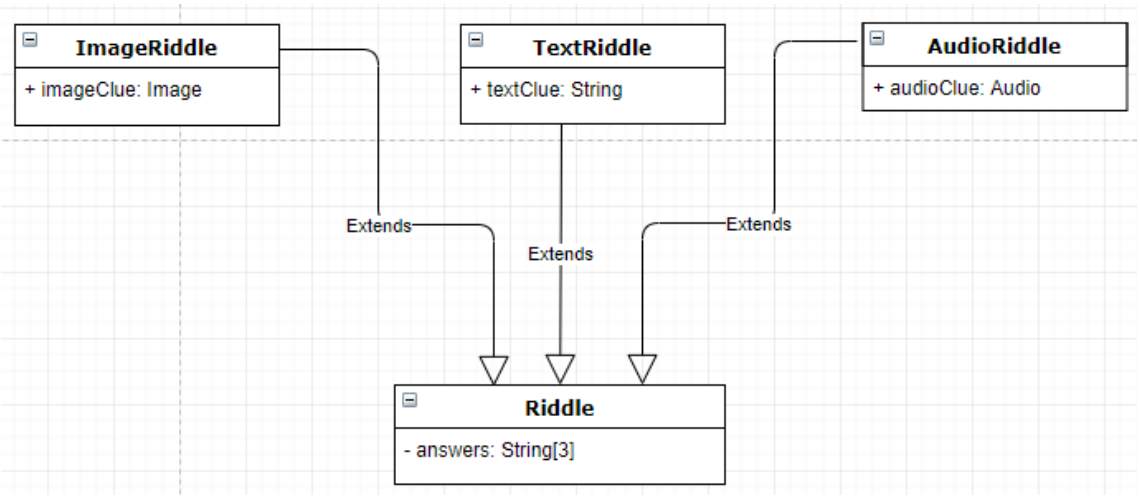
3. El diseño de la aplicación

Para el diseño de la aplicación, lo primero que hice fue pensar en las clases que serían necesarias para el funcionamiento de esta.

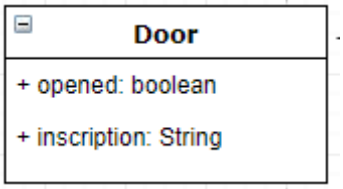
En todos los juegos que había diseñado anteriormente, para controlar al jugador utilizaba una clase que gestionaba, básicamente su movimiento. Así que decidí introducir dicha clase en el diseño, llamándola *Player*. Básicamente, incluye parámetros como su posición y su orientación.



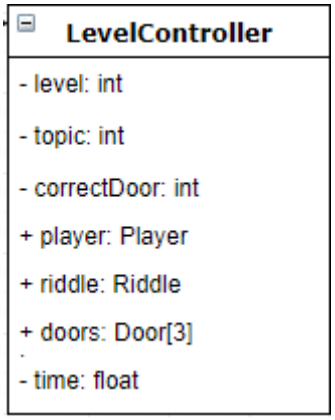
Por otra parte, para gestionar la lógica de los acertijos, lo que me pareció correcto fue crear una clase llamada *Riddle* y después crear otras clases que heredaran de ella, representando los diferentes tipos de acertijo. Un acertijo cualquiera contiene únicamente las posibles respuestas para su resolución. No es necesario indicar cual es la respuesta correcta, pues esta siempre estará en la misma posición. La subclase a la que pertenece un acertijo, indica el tipo de pista que encontraremos para resolverlo. Por tanto, tendremos tres subclases: *TextRiddle, ImageRiddle* y *AudioRiddle*. Cada una contiene un parámetro de un tipo diferente, que representa la pista.



También decidí añadir al diseño una clase que representara las puertas, ya que es necesario gestionar su apertura y clausura, además de introducir la inscripción que representa una de las respuestas del enigma que se plantea. Así pues, cree la clase *Door*, de la siguiente manera.



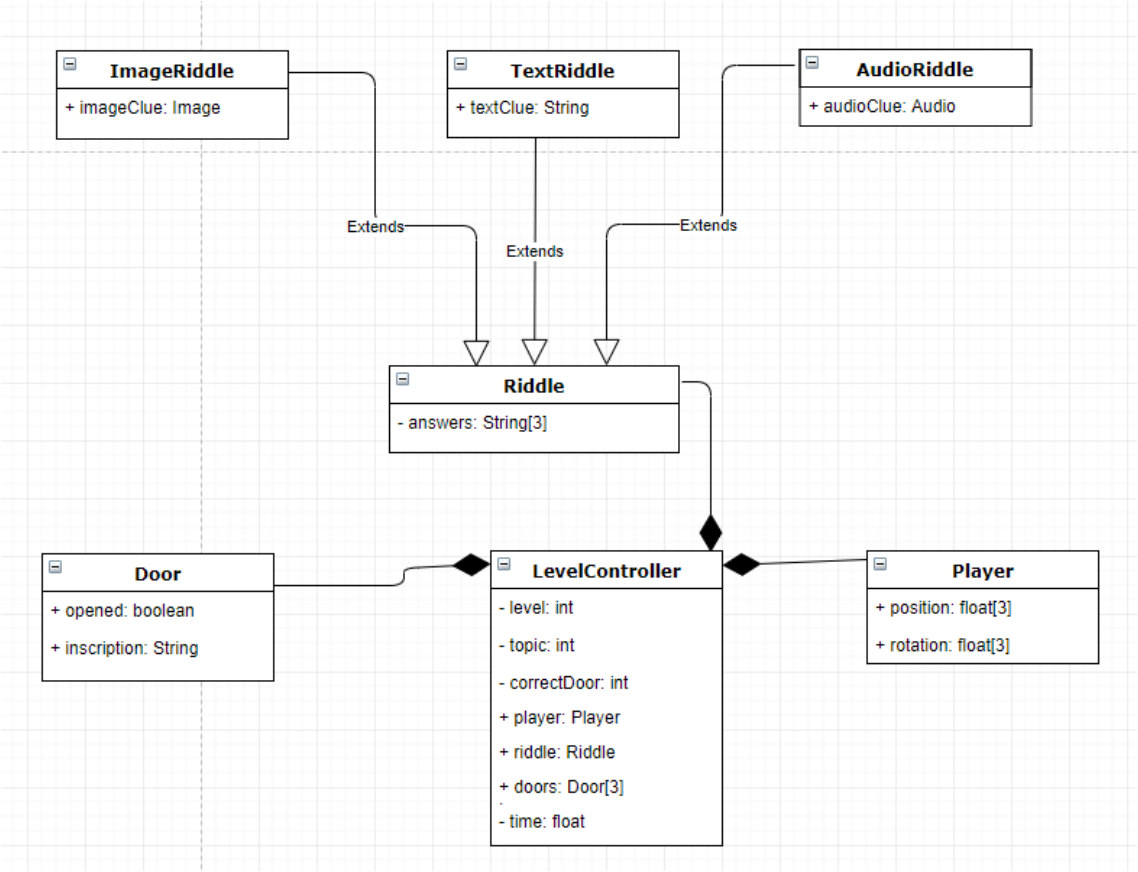
Finalmente, es necesario que cada vez que iniciemos un nuevo nivel, exista algo que se encargue de gestionar cuando superamos o perdemos el reto. Por ello, añadí al diseño la clase Game Controller, que incluye al jugador, el acertijo correspondiente al nivel y las tres puertas. También tiene otros parámetros como el número del nivel, el tema, cual es la puerta correcta (que se calcula de forma aleatoria) y el tiempo que nos queda para terminar.



De esta manera, la rutina de trabajo que sigue el controlador es la siguiente:

- Al empezar el nivel, toma el acertijo y sus tres respuestas y las coloca en las puertas de forma aleatoria, tomando en cuenta cual es la que contiene la respuesta correcta.
- En todo momento, el controlador está atento al tiempo. Si llega a 0, automáticamente se produce el final del nivel.
- También controla si la posición del jugador ha sobrepasado la de las puertas. Cuando esto sucede, se comprueba cual es la puerta que ha atravesado y finaliza en nivel, con victoria o con derrota.

Este sería, grosso modo, el diseño básico del funcionamiento del juego. Adjunto, en la siguiente imagen, el diagrama de clases completo.

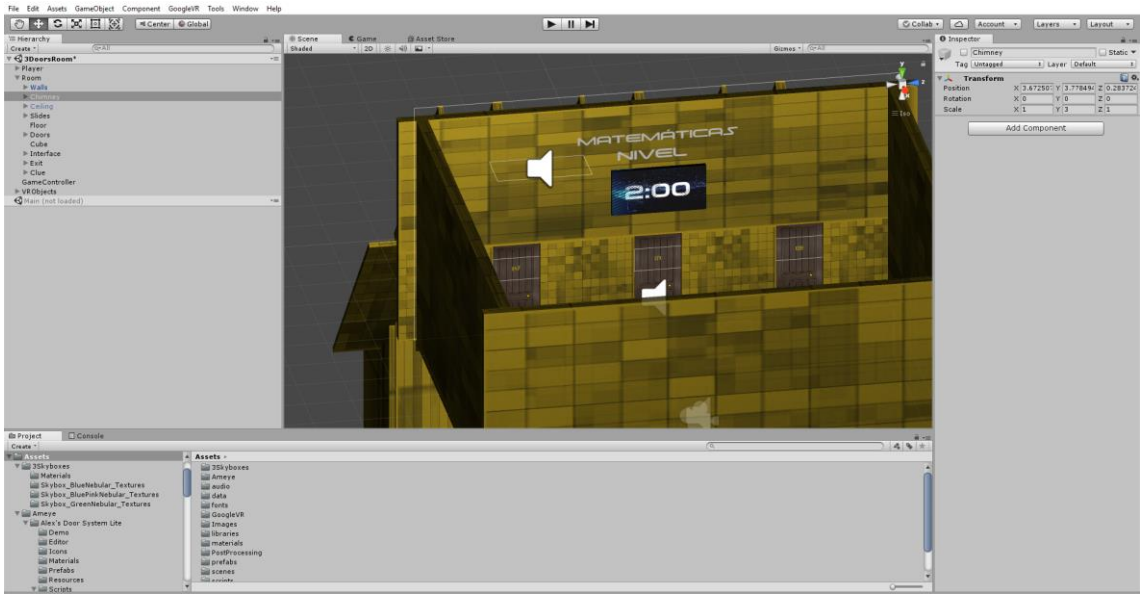

ImageRiddle
+ imageClue: Image
TextRiddle
+ textClue: String
AudioRiddle
+ audioClue: Audio
Extends
Extends
Extends
Riddle
- answers: String[3]
Door
+ opened: boolean
+ inscription: String
LevelController
- level: int
- topic: int
- correctDoor: int
+ player: Player
+ riddle: Riddle
+ doors: Door[3]
- time: float
Player
+ position: float[3]
+ rotation: float[3]

4. Herramientas utilizadas

4.1. La utilización de Unity en la elaboración del proyecto

Como he mencionado anteriormente, Unity ha sido la principal herramienta que he necesitado para confeccionar esta aplicación. Así que, considero de interés ofrecer lo que he podido aprender sobre este programa y, concretamente, sobre cómo lo he utilizado en mi proyecto.

Ya he contado que en el momento que comencé con el trabajo, estaba cursando la asignatura de Sistemas Multimedia. Hasta entonces había aprendido lo básico: la utilidad de las barras de herramientas, la manipulación de los objetos que introduzco en el entorno, la elaboración de scripts en C# de nivel básico… Dentro de todo esto, el tercer punto es el que más necesitaba desarrollar, puesto que los otros dos son muy intuitivos y con las explicaciones de Benoit en clase llegue a dominarlos de una forma lo suficientemente segura, de acuerdo con las necesidades previstas.

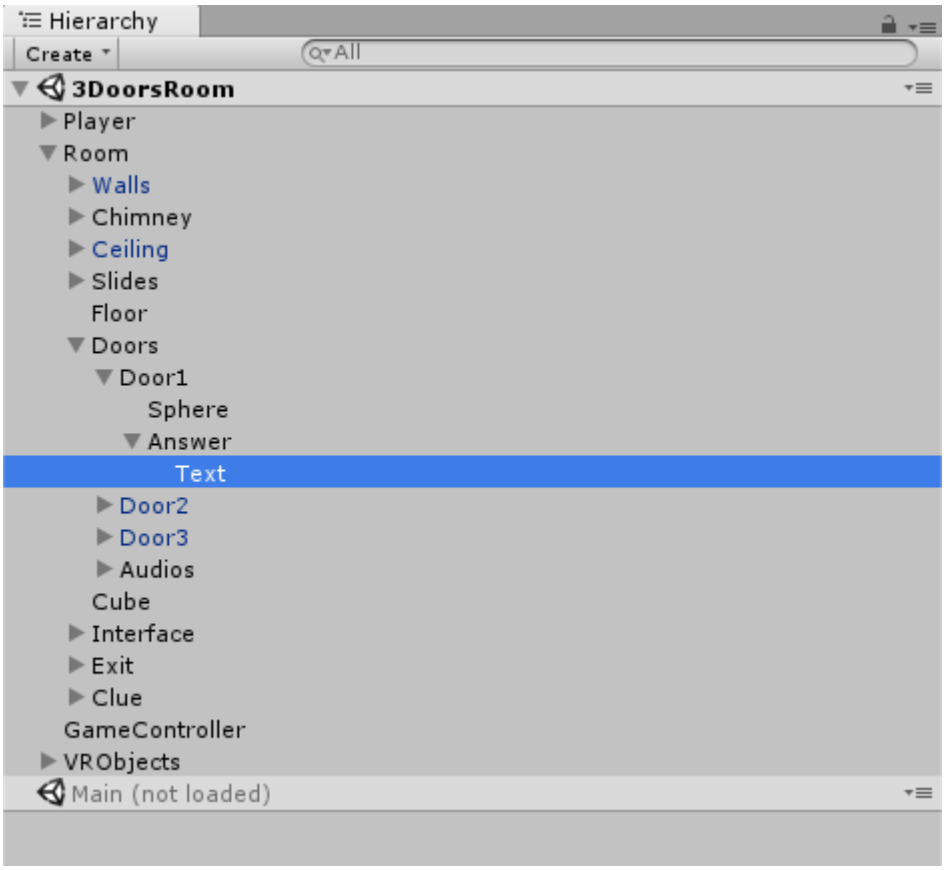


Una de las ventajas de Unity es que podemos personalizar el interfaz, colocando las diferentes ventanas prácticamente donde queramos. Contamos con una barra de herramientas en la parte superior, donde podemos realizar las típicas funciones de un editor corriente (abrir archivos, guardarlos, acceder a la configuración, a la ayuda…). Resulta también interesante conocer la función de las diferentes ventanas, así que explicaré las más importantes:

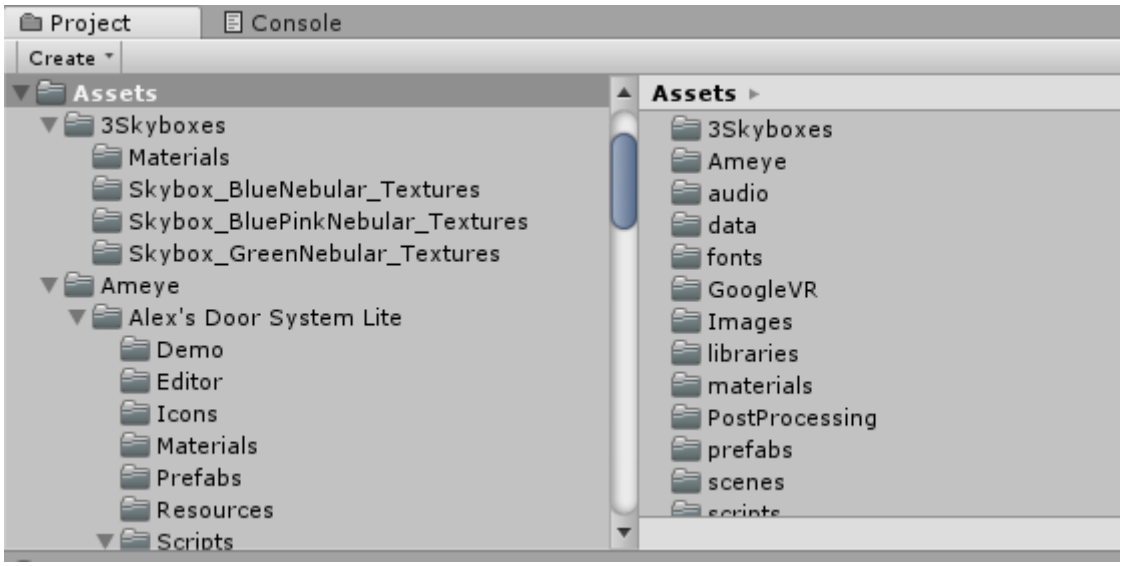
- **Scene:** Aquí tenemos la referencia visual de las escenas que estamos creando. Podemos navegar por ella haciendo uso del ratón y ayudándonos de la barra de herramientas que se encuentra justo debajo de la principal. Tenemos la

posibilidad de desplazarnos, mover los objetos, rotarlos o cambiar su tamaño directamente.

- **Hierarchy:** Aquí se muestran las diferentes escenas que estamos creando, pero en forma de jerarquía. Tenemos un despliegue de todos los objetos situados en la escena, mostrando cuales de ellos son hijos de otros. Por ejemplo, aquí observamos que el objeto *Room* incluye a su vez al objeto *Walls* y al objeto *Doors*, entre otros. Esta ventana es muy útil para establecer un orden y para que todos los objetos que son hijos de un mismo padre se comporten como una unidad.

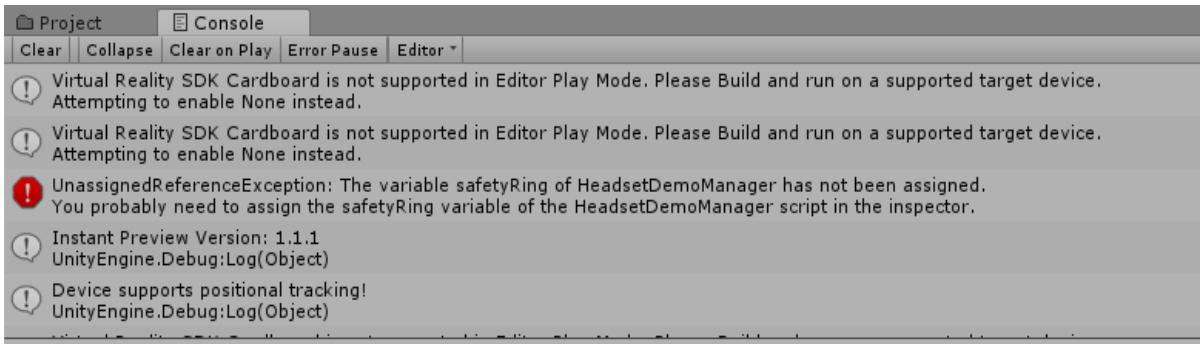


- **Project:** Aquí se muestra la jerarquía de ficheros del proyecto. Al crear un proyecto en Unity, se crea la carpeta Assets, donde guardaremos materiales, objetos prefabricados, texturas, escenas, librerías que importemos…

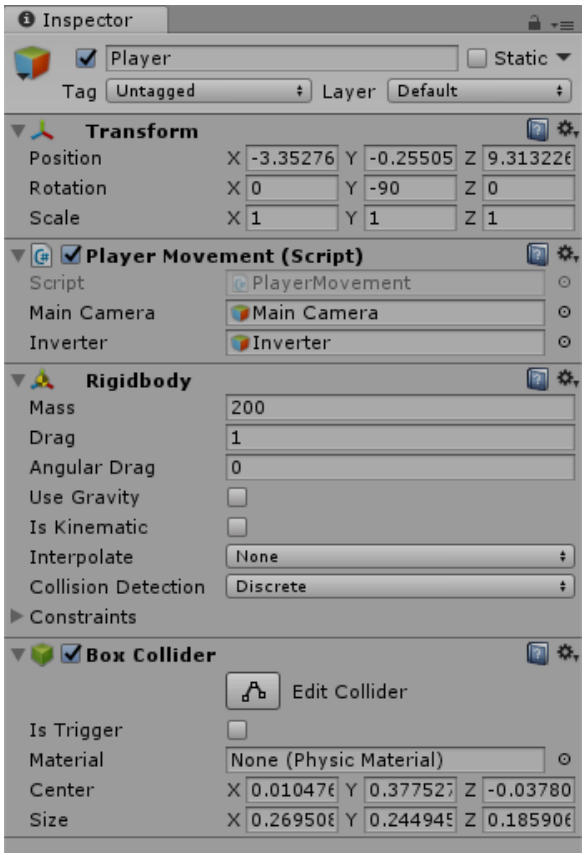


- **Game:** Aquí se mostrará la escena cuando ejecutemos el juego. Desde aquí cambiaremos la resolución y la escala de la pantalla.

- **Console:** Es la consola que nos mostrará errores, advertencias y mensajes en el momento de la ejecución.



- **Inspector:** Cuando seleccionamos un elemento en la escena, en la jerarquía o en el proyecto, aquí se mostrarán su nombre, sus componentes y otros parámetros. Podemos poner al objeto *Player* como ejemplo. Sus componentes son:
  - *Transform:* Todos los objetos en la escena tienen este componente, que indica su posición, su rotación y su escala.
  - *PlayerMovement:* Es un script que contiene la clase PlayerMovement. Añadiendo este componente le estamos diciendo a Unity que el objeto Player pertenece a esta clase y por tanto se comporta como tal.
  - *RigidBody:* Básicamente, le da un cuerpo físico al objeto, con cualidades como su masa.
  - *BoxCollider:* Los *colliders* permiten detectar colisiones de este objeto con otros que también tengan un *collider*. Es útil si quieres que suceda algo cuando dos objetos entren en contacto.



4.2 El Lenguaje C Sharp

C# (C Sharp) es el lenguaje que he utilizado para programar todos los scripts que contiene el proyecto, por lo que es uno de los puntos de mayor importancia. Se trata de un lenguaje orientado a objetos, lo cual favorece su utilización en Unity, al relacionar los objetos de las escenas con los del lenguaje.

4.3 Visual Estudio

Visual Estudio es un entorno de programación que Unity trae consigo y por consiguiente lo he utilizado en la creación los diferentes scripts que permiten el funcionamiento del juego. Al nivel al que lo he utilizado no tiene nada especial comparado con otros entornos de programación.

4.4 La librería de Google VR

Gracias a las aplicaciones de ejemplo que traía consigo la librería, conseguir descubrir, a grosso modo, cuál era su funcionamiento. Todas estas aplicaciones traían consigo una serie de elementos que contribuyen al mismo. Para comprenderlos antes hay que explicar que cuando ejecutamos el juego en el entorno de Unity, estamos simulando lo que veríamos, con ayuda de las gafas de realidad virtual, en nuestro dispositivo Android

- **GvrControllerMain:** Es el controlador del sistema. Se encarga de gestionar las acciones según el estado de los parámetros de la aplicación: la batería del dispositivo, la orientación del jugador, la aparición de errores…
- **GvrHeadset:** Es el elemento de conexión entre nuestro programa y la API que controla el dispositivo Android. Gracias a él, todo lo que funcione en el simulador de Unity, funcionará también en nuestro móvil o Tablet.
- **GvrInstancePreviewMain:** En el simulador, nos ofrece una vista de dos cámaras que se corresponderían con los dos visores de nuestras gafas de realidad virtual. Esto permite que tengamos en mente cómo se verá el juego cuando lo pongamos en marcha en el dispositivo Android.
- **GvrEditorEmulator:** Establece el modo de funcionamiento del emulador y permite editarlo. Por defecto, cuando ejecutamos la aplicación en Unity, tenemos la posibilidad de:
    - Hacer clic, que se correspondería con tocar la pantalla en el dispositivo. Esto puede generar una acción si el puntero genera por el objeto GvrPointerInputModule se encuentra sobre un elemento del entorno con el cual podemos interactuar.
    - Mover el ratón mientras mantenemos pulsado Alt o Ctrl, lo cual simula el giro de la cabeza en el entorno de realidad virtual. Las dos teclas (Ctrl y Alt) corresponden respectivamente a uno y dos ejes de giro, ya que solo podemos mover el ratón en un plano, mientras que el

movimiento de la cabeza se realiza en las tres dimensiones del espacio. Así pues, Ctrl simula el giro en el eje de la cámara, lo que conocemos como “roll”, mientras que Alt simula el eje en los otros dos ejes restantes, lo cual da lugar al “pitch” y al “yaw”.

- **GvrControlEventSystem:** Gestiona algunos de los eventos que suceden durante la ejecución. Entre otras cosas, contiene un GvrPointerInputModule, que se encarga de crear el puntero que sigue a nuestra cabeza cuando nos movemos con las gafas puestas. Se genera un rayo desde nuestra cámara principal y el puntero aparece allí donde el rayo colisiona con un elemento del entorno que posea un motor de colisiones.

5. La implementación

5.1 Las escenas

En este proyecto, existen tres escenas que intervienen en el desarrollo del juego. La primera escena es la escena del menú principal, llamada "Main", que contiene los siguientes elementos:

- Para empezar, contiene todos los elementos descritos en el apartado anterior, que permiten que el juego se muestre en modo VR.
- Al jugador, que realmente es la cámara capaz de girar sobre sí misma para poder observar el escenario.
- Un canvas de tipo world space que contiene diferentes paneles, representando los diferentes menús por los que podemos navegar en esta escena:
    - **StartMenu**: Es el menú que se muestra cuando se inicia la escena. En el tenemos un botón para empezar a jugar, otro para ver el tutorial y otro para salir del juego.
    - **ModeMenu**: Se muestra cuando seleccionamos la opción de jugar en el menú anterior. Desde aquí podemos seleccionar el modo normal o el modo invertido, o bien podemos regresar al menú start.
    - **TopicMenu**: Después de elegir el modo, tendremos que elegir el tema de los acertijos, por lo que tendremos un botón para elegir cada tema. También se permite regresar al menú anterior.

La segunda escena es "Tutorial", cuya finalidad es mostrar al jugador el funcionamiento de la aplicación para una correcta utilización y una mejor experiencia. El tutorial contiene:

- Al igual que la escena anterior, al jugador con las mismas capacidades.
- También contiene una serie de paneles con textos e imágenes que nos ayudan a comprender el funcionamiento del juego.
- Botones de avanzar y retroceder, que nos permiten navegar entre los diferentes paneles y volver al menú principal.

La tercera escena, llamada “3doorsRoom”, es el juego en sí, al cual accedemos una vez hemos elegido el modo de juego y el tema. Vamos a enumerar los diferentes elementos que contiene.

- Al igual que en la escena anterior, se incluyen los elementos necesarios para que la realidad aumentada funcione.
- La habitación, que a su vez está formada por: las paredes, el suelo, el techo, la chimenea (un tubo encima del techo por el cual el jugador entra en la habitación), las puertas y los túneles que se encuentran detrás de las mismas.
- El jugador, que inicialmente se encuentra en el interior de la chimenea y cae en la habitación. Incluye la cámara y un objeto llamado inverter, que tendrá utilidad cuando juguemos el modo invertido.
- El controlador, un objeto que contiene un script de vital importancia para el funcionamiento del juego, el cual será explicado con detalle más adelante.
- El interfaz situado justo encima de las puertas, donde encontraremos el cronómetro, el tema elegido y el nivel que tratamos de superar.

- El botón de escape. Al pulsarlo, abre un panel que nos pregunta si queremos salir del juego. Es lo que nos permite abandonar el juego si nos cansamos de jugar.

- La pista, que realmente está dentro de la habitación. Incluye los diferentes elementos necesarios para que esta se pueda mostrar en función del tipo al que pertenezca: un texto y varios audios e imágenes.

5.2 La construcción del escenario principal

Después de haber comprendido el funcionamiento de los elementos descritos en el apartado anterior, lo que hice fue tomar una de las aplicaciones que traía como ejemplo la librería de Google VR como base para construir la escena principal. Esto incluye, al jugador, que simplemente esta formado por una cámara, y los elementos ya mencionados.

No quería abusar de los objetos que se pueden descargar desde el Asset Store, principalmente porque me gusta el diseño gráfico. Tanto las paredes como las puertas de la habitación son cubos 3D aplanados para que tengan el grosor correcto. Cada uno tiene un "Collider" y un "RigidBody" que evitan que nuestro jugador pueda atravesarlo.

Me descargué imágenes para darles a los elementos un aspecto y texturas más realistas. Además, en las puertas coloqué un pomo que es simplemente una esfera con una textura metálica básica.


MATEMÁTICAS
NIVEL
2:00
Iso

5.3 Implementación de clases y funciones

En este apartado explicaré detalladamente todas las funciones que he implementado para el correcto funcionamiento de la aplicación.

5.3.1 Interacción con los objetos

Es lo primero que se debe explicar para poder entender los apartados que vienen a continuación. Nuestra aplicación tenía un problema bastante importante. Se trataba de que la única forma de interactuar con los elementos del escenario era tocar la pantalla (hacer clic en el simulador). ¿Pero, cómo vamos a tocar la pantalla si nuestro dispositivo Android se encuentra introducido en las gafas de realidad virtual?

Estaba claro que teníamos que idear otro modo de interacción. Lo primero que se nos ocurrió, fue utilizar un control remoto; utilizar otro dispositivo el cual, mediante una aplicación, estableciera una conexión con el que está ejecutando el juego y permita que, al tocar la pantalla de uno, se simule que se ha tocado la del otro. Terminamos descartando esta idea por dos razones:

- Era un impedimento necesitar dos dispositivos para ejecutar la aplicación.
- Las aplicaciones que encontramos eran difíciles de comprender, además de que su funcionamiento no estaba muy pulido.

Encontramos otra posible solución: utilizar un sistema en el cual, cuando se mira fijamente un elemento durante un tiempo determinado, se simula que hemos tocado dicho elemento. Básicamente, cuando el puntero "entra" en el objeto, si existe alguna forma de interactuar con él tocándolo, se inicia un cronómetro. Si el puntero sale del elemento antes de que finalice el tiempo, no ocurre nada, pero en caso contrario, se lleva a cabo la acción correspondiente a tocar dicho elemento.

Así pues, todos los elementos con los cuales se puede interactuar de esta manera tienen anexado el script *ObjectsController.cs,* que implementa la clase *ObjectsController*. Dejo aquí el código para explicarlo paso a paso.

```
public class ObjectsController : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
{

    // Use this for initialization

    private float timer;
    private bool timerActive;

    void Start () {
        timer = 1;
        timerActive = false;
    }

    // Update is called once per frame
    void Update () {
        if (timerActive)
        {
            timer -= Time.deltaTime;
            if (timer <= 0)
            {
                ExecuteEvents.Execute(gameObject,
                    new PointerEventData(EventSystem.current),
                    ExecuteEvents.pointerClickHandler);
                timer = 1;
                timerActive = false;
            }
        }
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        timerActive = true;
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        timerActive = false;
        timer = 1;
    }
}
```

Solo utilizamos dos variables: *timer* indica el tiempo que falta para que la simulación del clic se active. Hemos seleccionado 1 segundo como tiempo inicial. *timerActive* indica si dicho tiempo está corriendo. Lógicamente, en un primero momento esta variable se inicializa como falsa.

En la función *Update(),* que se ejecuta en cada frame, comprueba si el contador de tiempo está activo. En caso afirmativo, a la variable *timer* le restamos el tiempo que ha transcurrido en ese frame y después comprobamos si ha llegado a 0. Si es así, ejecutamos la acción correspondiente al clic y después reiniciamos el timer.

La función *OnPointerEnter*, se ejecuta cuando el puntero se coloca sobre el objeto. En ese caso, simplemente tenemos que activar el timer. Mientras que la función *OnPointerExit*, se ejecuta cuando el puntero deja de estar sobre el objeto. Ante lo cual debemos reiniciar el timer.

En los menús del juego tenemos diferentes botones, los cuales utilizan un sistema similar, mediante el script *ButtonsController.cs*. Se diferencia de *ObjectsController* en que las funciones para pulsar los botones son diferentes que las de tocar los objetos.

5.3.2 El interfaz

El juego contiene un interfaz donde se muestran la mayor parte de los elementos UI. Se trata de un canvas. Según la documentación de Unity, un Canvas es "*el área donde todos los elementos UI deben estar*", refiriéndose a elementos como textos, botones o paneles. Los Canvas tienen varios modos de renderización:

- Screen Space (Espacio de la pantalla): Este modo de renderización coloca elementos UI en la pantalla mostrada en la parte superior de la escena. Si el tamaño de la pantalla es modificado o cambia la resolución, el Canvas va a automáticamente cambiar el tamaño para que coincida. A su vez, puede basarse en la pantalla en sí o en una cámara en concreto.
- World Space (Espacio del mundo): Los elementos UI están en una posición fija del espacio y si la cámara se mueve, los veremos desde otra perspectiva.

Pero resulta que tenemos un problema, la librería VR de Google aún no está preparada para mostrar este tipo de interfaces y solo podemos ver los canvas de tipo world space. Por tanto, utilizaremos este sistema.

Los elementos del interfaz son los siguientes:

- El tiempo restante, que es de dos minutos al comienzo del nivel. Si este tiempo se agota, el jugador no podrá resolver el acertijo.
- El tema o asignatura escogido.
- El nivel actual en el que se encuentra el jugador.
- El menú de abandono: Para activarlo solo debemos tocar el botón Exit que se encuentra en la pared trasera. Contiene un botón para abandonar y otro para seguir jugando (ya mostrado en otro apartado).

- El panel de derrota: Incluye el mensaje que nos dice que hemos perdido y un botón que nos devuelve al menú principal.

- El panel de victoria: Similar al anterior, pero el mensaje nos dice que hemos superado el reto.

5.3.3 El menú principal

Como todo programa, esta aplicación cuenta con un menú principal. Cuando trabajamos en realidad virtual, para crear este menú tenemos que utilizar un *canvas* de tipo *world space*. Se encuentra situado delante de nosotros cuando iniciamos la aplicación, pero podemos girar la cabeza y dejar de verlo. El menú principal tiene simplemente tres botones: uno para empezar a jugar, otro para mostrar el tutorial y otro para salir.

Si elegimos Empezar, podremos elegir entre los dos modos básicos de juego: normal e invertido. El modo normal no tiene ningún misterio, mientras que el invertido tiene la particularidad de que, al girar la cabeza de izquierda a derecha, la cámara se moverá en dirección contraria. El objetivo de este modo es que los chavales trabajen no solo sus conocimientos académicos, si no también sus habilidades motrices y su coordinación.

Una vez elegido el modo de juego, solo nos falta seleccionar la asignatura que queremos trabajar y el juego comenzará.

5.3.4 El jugador

En un principio, el jugador solo estaba formado por la cámara, que son sus ojos en todo momento. Su única capacidad de acción era girar sobre sí mismo para poder observar la habitación, pero mi intención era que se pudiera recorrer libremente. Por tanto, comencé con la creación del script llamado *PlayerMovement.cs*, que implementa la clase *PlayerMovement*. Mi idea era que al tocar la pantalla cuando el puntero miraba al suelo, el jugador se desplazase a esa posición. Pero no quería que lo hiciese de manera inmediata, si no que se simulara que el jugador camina.

Vamos a mostrar primero cuales son las variables que utilizaremos en este script:

```
public GameObject mainCamera;
public GameObject inverter;
private float invertRotation;
private float cameraRotation;
private bool inverted;

private Vector3 goal;
private Vector3 direction;
```

- *MainCamera* es un objeto que representa la cámara principal.

- *Inverter* es un objeto que contiene a la cámara. Su utilidad es la de invertir el giro de esta en el caso de que juguemos el modo invertido.
- *invertRotation* y *cameraRotation* son parámetros que también nos servirán cuando utilicemos el modo invertido.
- *Inverted* es una variable booleana que indica si estamos jugando el modo invertido o no.
- El vector *goal* indica la posición objetivo a la que debe moverse el jugador en un momento dado.
- El vector *direction* indica, valga la redundancia, la dirección en la que se debe mover el jugador para alcanzar la posición objetivo.

Utilizamos la función *Update()*, la cual es llamada automáticamente en cada frame. En ella, primero realizamos la rotación inversa correspondiente en el caso de que se esté probando el modo invertido (En el siguiente apartado se explica esto con detalle). También comprobamos si la posición del jugador es aproximadamente la misma que la posición objetivo. Si no es así, esperamos 1 décima de segundo y movemos al jugador de manera que se acerque a donde nosotros queremos.

```
// Update is called once per frame
public void Update()
{
    if (inverted)
    {
        invertRotation = mainCamera.transform.localRotation.eulerAngles.y - cameraRotation;
        inverter.transform.Rotate(0, -2 * invertRotation,0);
        cameraRotation = mainCamera.transform.localRotation.eulerAngles.y;
    }

    if (Mathf.Abs((transform.position - goal).magnitude) > 0.1f)
    {
        StartCoroutine(Wait());
        transform.position += direction;
    }
}

IEnumerator Wait()
{
    yield return new WaitForSeconds(0.1f);
}
```

De esta forma se consigue que el jugador se mueva poco a poco hacia la posición indicada, en lugar de hacerlo de golpe.

Para actualizar las variables *goal* y *direction*, utilizamos la función *SetGoal*. Le pasamos como argumento la posición del puntero que vemos en pantalla. La segunda línea de la función sirve para que el jugador se mueva solo en el plano del suelo.

```
public void SetGoal(Vector3 pointerPosition)
{
    goal = pointerPosition;
    goal.y = transform.position.y;
    direction = (goal - transform.position).normalized * 0.1f;
}
```

¿Y en qué momento llamamos a esta función? A priori, parece que lo que tenemos que hacer es incorporar en el suelo el script *ObjectsController* descrito en el apartado anterior y decirle a Unity que queremos llamar a *SetGoal* cuando hagamos clic sobre él. Pero realmente, surge un problema: queremos que el jugador se mueva únicamente si miramos a un punto fijo del suelo, pero tal como está construido el script, si movemos la cabeza mirando al suelo, al cabo de 1 segundo nos moveremos hacia donde estemos mirando.

De modo que crearemos un nuevo script para implementar la clase *FloorController*, que hereda de *ObjectsController*.

```
public class FloorScript : ObjectsController, IPointerClickHandler, IPointerEnterHandler {

    public GameObject pointer;
    private Vector3 pointerPosition;
    private Vector3 goalPosition;
    public GameObject player;

    // Update is called once per frame
    public new void Start()
    {
        base.Start();
        pointerPosition = pointer.GetComponent<GvrReticlePointer>().GetPointerPosition();
        goalPosition = pointerPosition;
    }
```

```
    public new void Update()
    {
        if (timerActive)
        {
            if ((goalPosition - pointerPosition).magnitude > 0.5)
            {
                ExecuteEvents.Execute(gameObject,
                    new PointerEventData(EventSystem.current),
                    ExecuteEvents.pointerExitHandler);
                ExecuteEvents.Execute(gameObject,
                    new PointerEventData(EventSystem.current),
                    ExecuteEvents.pointerEnterHandler);
            }
            else
            {
                timer -= Time.deltaTime;
                if (timer <= 0)
                {
                    ExecuteEvents.Execute(gameObject,
                    new PointerEventData(EventSystem.current),
                    ExecuteEvents.pointerClickHandler);
                    timer = 1;
                }
            }
        }
        pointerPosition = pointer.GetComponent<GvrReticlePointer>().GetPointerPosition();
    }

    public new void OnPointerEnter(PointerEventData eventData)
    {
        goalPosition = pointerPosition;
        Debug.Log(pointerPosition);
        timerActive = true;
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        player.GetComponent<PlayerMovement>().SetGoal(pointerPosition);
    }

}
```

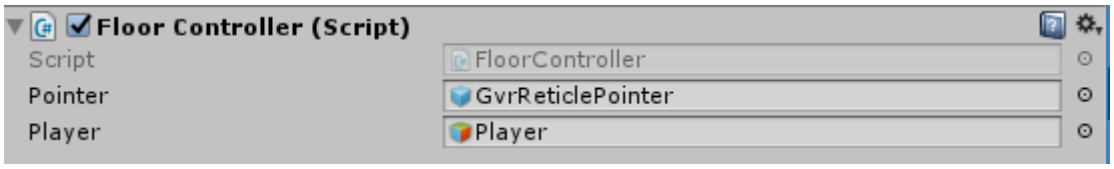
Como se puede apreciar, añadimos varias variables:

- *pointer*: Es el objeto que representa el puntero que queremos seguir.
- *pointerPosition*: Será en todo momento la posición del puntero.
- *goalPosition*: La posición a la que queremos acceder.
- *player*: Un objeto que representa al jugador.

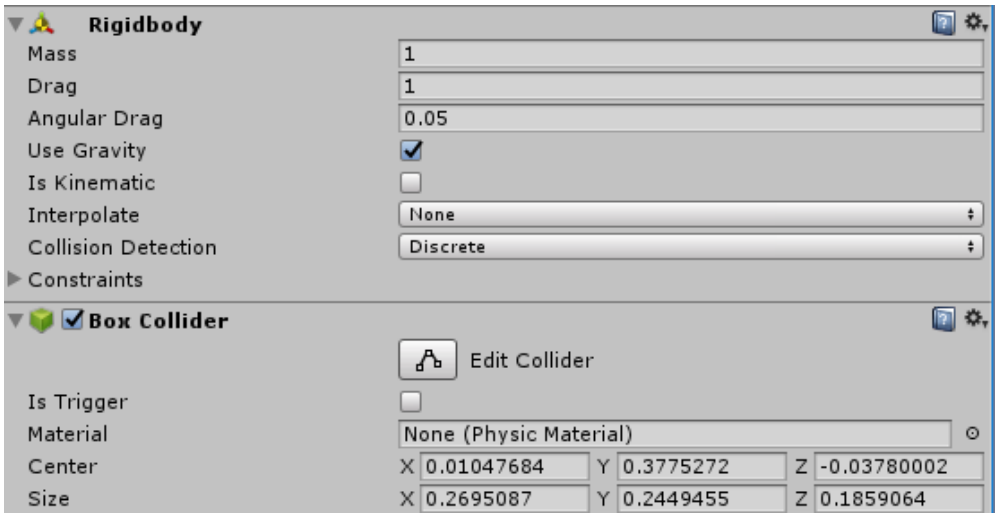
Si observamos la función *Update*, vemos que en caso de que el *timer* se encuentre activo, comprobamos que la posición del puntero no haya cambiado demasiado respecto a la última posición objetivo que hemos guardado. En caso de que el cambio sea suficiente, tomamos una nueva posición objetivo (para ello simulamos que el puntero ha entrado y salido del objeto) y reiniciamos el contador de tiempo.

Ahora, en la función *OnPointerEnter*, actualizamos la posición objetivo. Aprovechamos para crear la función *OnPointerClick*, que se ejecuta cuando hacemos clic sobre el elemento y le decimos que llame a *SetGoal*, pasándole como argumento nuestra variable *pointerPosition*.

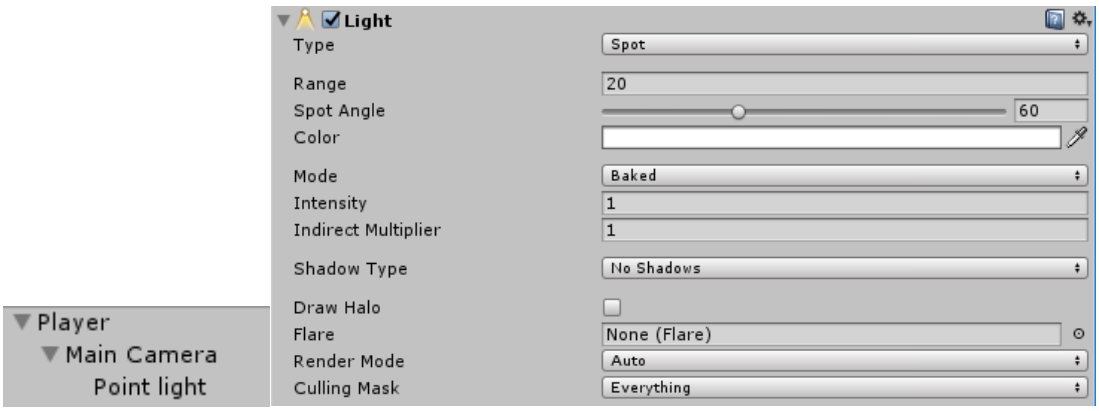
Pero para que todo esto funcione, tenemos que decirle a Unity que objetos representan las variables *pointer* y *player*. Afortunadamente, en este motor resulta muy sencillo y solo necesitamos arrastrar los objetos correspondientes.



Al permitir el movimiento del jugador, era necesario añadir un Collider y un RigidBody para evitar que este atravesara los elementos físicos del escenario. Utilizamos la gravedad, pues recordemos que, para pasar de nivel, el jugador debe lanzarse al vacío, jugándosela a caer en buen lugar.



Por último, hemos añadido a la jerarquía un foco que sigue a la cámara y que permite al jugador iluminar la habitación allá donde mira.



Hemos establecido que el tipo de luz sea un foco y hemos jugado con los parámetros de rango, que indica la máxima distancia a la que llega la luz del foco e intensidad, que establece cómo de potente es dicha luz.

5.3.5 El modo invertido

Ya hemos comentado que el juego tiene un modo invertido. Consiste en invertir el sentido del movimiento cuando giramos la cabeza en torno al eje vertical, con el objetivo de trabajar también las habilidades motrices. El motivo de invertir solo este eje se explica en el apartado de "Problemas encontrados".

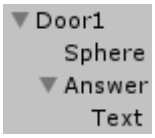
Como se ha mencionado, es en la clase del jugador donde efectuamos el giro invertido, concretamente en la función *Update*.

```
if (inverted)
{
    invertRotation = mainCamera.transform.localRotation.eulerAngles.y - cameraRotation;
    inverter.transform.Rotate(0, -2 * invertRotation,0);
    cameraRotation = mainCamera.transform.localRotation.eulerAngles.y;
}
```

Primero asignamos a la variable *invertRotation* (float) la diferencia entre el valor de rotación actual en el eje vertical y el mismo valor del frame anterior. Es decir, *invertRotation* indica cuanto hemos girado en el último frame. Entonces, giramos el objeto *inverter* esa cantidad de grados en el eje Y, multiplicada por -2. Si en un frame nos hemos girado, por ejemplo, 10 grados a la derecha, lo compensamos girándonos 20 grados a la izquierda. De esta manera, la rotación total será de 10 grados a la izquierda, justo lo que queríamos. La última instrucción se encarga de guardar la rotación a actual de la cámara en la variable *cameraRotation*, para utilizarla en el siguiente frame.

5.3.6 El funcionamiento de las puertas

Para explicar el funcionamiento, primero debemos explicar la estructura de las puertas. Cada puerta consta de varios elementos:



Sphere es una esfera que representa el pomo de la puerta. Su distinción es necesaria porque es el elemento que queremos mirar fijamente para que la puerta se abra.

Answer es un Canvas. Elegimos el modo World Space porque queremos que el texto que contiene la puerta, que representa la inscripción, se mantenga pegado a ella.

Lo que nos interesa ahora es implementar una forma de que la puerta se abra. Lo primero que hacemos es añadir el script ObjectsController al pomo, ya que queremos interactuar con él. Una vez hecho esto, vamos a crear un nuevo script para implementar la clase DoorBehaviour.

```
public class DoorBehaviour : MonoBehaviour{

    // Use this for initialization
    public AudioSource openAudio;
    public AudioSource closeAudio;
    private Vector3 openingRotation;
    private Vector3 openingTranslation;
    private Vector3 closingTranslation;
    private bool opened;

    void Start () {
        openingRotation = new Vector3(0, 1, 0);
        openingTranslation = new Vector3(-0.045f, 0, -0.045f) ;
        closingTranslation = new Vector3(0.045f, 0, 0.045f);
        opened = false;
    }

    public void KnobTouch()
    {
        if (opened)
        {
            StartCoroutine(Close());
        }
        else
        {
            StartCoroutine(Open());
        }
        opened = !opened;

    }


    public IEnumerator Open()
    {
        openAudio.Play();
        for (int i = 0; i < 90; i += 10)
        {
            transform.Translate(openingTranslation, Space.World);
            transform.Rotate(openingRotation, -9);
            yield return new WaitForSeconds(0.05f);
        }
    }

    public IEnumerator Close()
    {
        closeAudio.Play();
        for (int i = 0; i < 90; i += 10)
        {
            transform.Translate(closingTranslation, Space.World);
            transform.Rotate(openingRotation, 9);
            yield return new WaitForSeconds(0.05f);
        }
    }
}
```

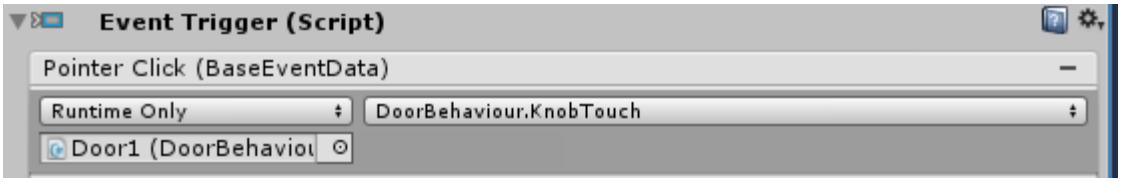
Empecemos por explicar las variables que se van a utilizar:

- *openAudio* y *closeAudio:* Son dos variables de tipo *AudioSource*. Esto quiere decir que tienen un archivo de audio anexado, en este caso de formato mp3 y podemos controlar su reproducción. Como su nombre indica, estos audios re reproducirán cuando abramos y cerremos la puerta, respectivamente.
- *openingRotation:* Es el eje sobre el que gira la puerta cuando se abre y se cierra. En la función *Start* se observa que se inicializa con el eje Y.
- *openingTranslation* y *closigTranslation:* Son dos vectores que indican el desplazamiento que necesita la puerta cuando se abre y se cierra para que las bisagras sigan pegadas al marco de la puerta.
- *Opened:* Indica en todo momento si la puerta está abierta o no lo está. Se inicializa como falso.

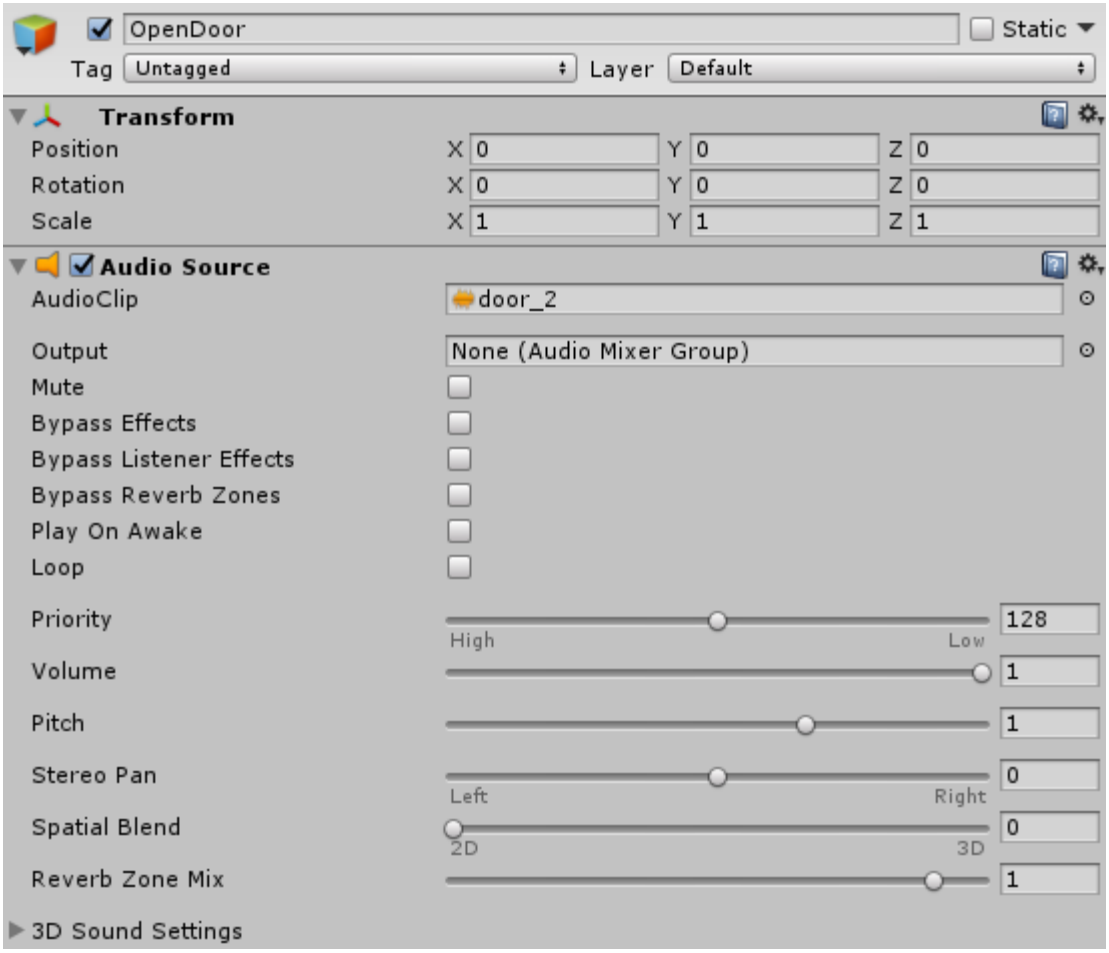
La función *KnobTouch* se encarga de llamar a las otras dos funciones de tipo *IEnumerator: Open*, si la puerta está cerrada y *Close* si la puerta está abierta. *Open*, reproduce el audio correspondiente y mientras tanto, va abriendo la puerta poco a poco, esperando 5 centésimas de segundo en cada iteración. *Close* ejecuta el mismo proceso, pero a la inversa.

Entonces, lo lógico es pensar que simplemente tenemos que conseguir que cuando toquemos el pomo, llamemos a la función *KnobTouch* para que la puerta se abra o se cierre. Y esto es cierto, pero antes, cuando hemos explicado el funcionamiento del movimiento del jugador, también hemos trabajado con la clase *ObjectsController* y hemos llegado a la conclusión de que necesitábamos la herencia para que funcionase. Ahora podría parecer que también la necesitamos, pues aún existen otros elementos con los que podemos interactuar, para los cuales, la función *OnPointerClick* será diferente a la del pomo de la puerta. ¿Tenemos que construir una subclase por cada elemento diferente que utiliza la clase *ObjectsController*?

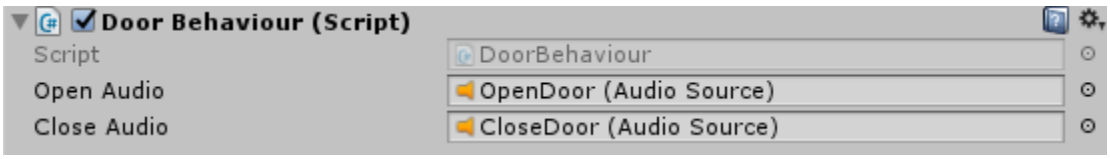
Afortunadamente, Unity nos libera de este trabajo gracias a los componentes de tipo *Event Trigger.* Estos nos permiten gestionar las acciones que sea realizan cuando colocamos tocamos o simplemente colocamos un cursor sobre el objeto. Así que utilizamos un *Event Trigger* sobre el pomo y le decimos que cuando se toque, se llame a la función *KnobTouch* correspondiente a la puerta a la que pertenece, tal y como se muestra en la siguiente imagen.



Los objetos de tipo *AudioSource* tenemos que crearlos en el interfaz de Unity y asignarles el archivo de audio correspondiente.



Después, solo tenemos que arrastrar dichos objetos allí donde utilicemos la clase *DoorBehaviour*.



5.3.7 La lógica de los acertijos

Como se ha mencionado anteriormente, para poder salir de la habitación, tenemos que resolver un acertijo. He creído conveniente implementar una clase que represente a dichos acertijos, a la cual he llamado *Riddle*. En la siguiente imagen os muestro el código de esta.

```
public class Riddle
{
    private string[] answers;


    public Riddle(string answer1, string answer2, string answer3)
    {
        answers = new string[] { answer1, answer2, answer3 };
    }

    public string GetAnswer(int number)
    {
        return answers[number];
    }
}
```

Como se puede apreciar, el único parámetro que contiene la clase son las respuestas, representadas por la variable *answers*. El constructor tiene como variables de entrada tres cadenas de caracteres que se introducen en el vector. Cabe destacar que siempre debemos introducir la respuesta correcta en la tercera posición. Introducimos la función *GetAnswer*, que nos permitirá acceder a la respuesta de la posición que le pasemos como argumento.

Por tanto, no tenemos, por el momento, información a cerca de la pista que nos ayuda a resolver el enigma. Esto es así porque, tal y como se ha explicado, tenemos tres clases de acertijo y en cada uno de ellos, la pista será de un tipo diferente. Así que utilizamos la herencia para crear dichas clases de acertijo.

```
public class TextRiddle: Riddle
{
    private string textClue;

    public TextRiddle(string question, string answer1, string answer2, string answer3):
        base(answer1, answer2, answer3)
    {
        textClue = question;
    }

    public string GetQuestion()
    {
        return textClue;
    }
}

public class ImageRiddle : Riddle
{
    private Image imageClue;

    public ImageRiddle(Image image, string answer1, string answer2, string answer3) :
        base(answer1, answer2, answer3)
    {
        imageClue = image;
    }

    public Image GetImage()
    {
        return imageClue;
    }
}

public class AudioRiddle : Riddle
{
    private AudioSource audioClue;

    public AudioRiddle(AudioSource audio, string answer1, string answer2, string answer3) :
        base(answer1, answer2, answer3)
    {
        audioClue = audio;
    }

    public AudioSource GetAudio()
    {
        return audioClue;
    }
}
```

En cada una de las subclases introducimos la variable de la pista, que puede ser un texto, una imagen o un audio. En cada caso, simplemente creamos el constructor y la función que nos permitirá acceder a la pista, que siempre es una variable privada.

5.3.8 El Controlador

En el escenario, tendremos un objeto de vital importancia que se encargará de gestionar la lógica del juego. Será de la clase *GameController* y se generará una instancia cada vez que comience la escena, o dicho de otro modo, cada vez que iniciemos un nuevo nivel:

```
public class GameController : MonoBehaviour {
```

Conviene explicar paso a paso todas las funciones que cumple este objeto y como siempre, vamos a comenzar mostrando las variables utilizadas. Como la cantidad de variables es grande, las vamos a dividir en varios grupos:

- Variables lógicas

```
private int level;
public int lastLevel;
private int topic;
public Riddle myRiddle;
private Riddle[,] riddles;
private int correctDoor;
private float leftLimit;
private float rightLimit;
```

  - *Level:* Es un entero que indica el nivel en el que nos encontramos o, en otras palabras, cuantos enigmas hemos resuelto ya.
  - *lastLevel:* Indica el número de enigmas que debemos resolver para terminar el juego y ser libres. Es una variable pública porque así podemos modificarla fácilmente desde el interfaz de Unity.
  - *Topic:* Señala el tema de los acertijos. Por ejemplo, el número 1 significa que los acertijos son de matemáticas. Después mostraré un script donde se guardan datos de este tipo.
  - *MyRiddle:* Es un objeto de la clase *Riddle* que indica el acertijo que se debe resolver en el nivel en el que nos encontramos.
  - *Riddles:* Es el conjunto de todos los posibles acertijos. Se trata de una matriz en la que la fila indica el tema y la columna el nivel en el que nos encontraremos con cada uno de los enigmas.
  - *correctDoor:* Indica cuál de las puertas debemos atravesar para resolver el enigma. Más adelante veremos que se genera de manera aleatoria.
  - *leftLimit y rightLimit:* El nivel termina cuando el jugador atraviesa cierto plano del espacio. Dependiendo de por donde lo atraviese, ganará o perderá, lo cual depende de cuál es la puerta correcta. Estas dos variables limitan el espacio por donde debe atravesar el plano el jugador para salir victorioso.

- Variables que son elementos físicos del escenario:

```
public GameObject player;
public GameObject[] doors;
public GameObject clue;
public Material wallMaterial;
public Camera cam;
public GameObject inverter;
public GameObject room;
public MeshRenderer reticlePointer;
```

  - *Player:* Como su nombre indica, es el jugador.
  - *Doors:* Un array de objetos que almacena las tres puertas de la habitación
  - *Clue:* Representa la pista del acertijo. Contiene un texto y los archivos de imagen y audio necesarios. Nos interesa dar aleatoriedad a su posición para que no sea tan fácil de encontrar.
  - *wallMaterial:* El material de las paredes. Su única finalidad es poder modificar el aspecto de la habitación según el nivel, para que el juego no sea tan monótono.
  - *Cam:* La cámara que serán los ojos del jugador.
  - *Inverter:* El objeto que ya hemos explicado antes para el modo invertido.
  - *Room:* La habitación en su conjunto.
  - *reticlePointer:* Es el puntero que sigue a la cámara. Más adelante veremos porque es necesario.
- Variables del interfaz

```
private float chronometer;
public Text timerText;
public Text topicText;
public Text levelText;
public GameObject loosingMenu;
public GameObject winningMenu;
public GameObject exit;
```

  - *Chronometer:* El jugador tiene 2 minutos para resolver el enigma y salir de la habitación. El tiempo restante se representa con esta variable.
  - *timerText:* Es el texto que refleja el valor de la variable chronometer.
  - *TopicText:* Texto que indica cual es el tema elegido.
  - *levelText:* Indica el nivel en el que nos encontramos.
  - *loosingMenu:* Es el menú que aparece automáticamente cuando el jugador termina el nivel sin éxito.
  - winningMenu: El menú que aparece cuando el jugador ha superado todos los niveles.

- *Exit:* Es un panel que se muestra cuando el jugador interactúa con el botón que se encuentra en la pared trasera, lo cual indica que quiere abandonar.

Como es sabido, cuando se crea un objeto MonoBehaviour, automáticamente se llama a la función Start, la cual aprovecharemos para preparar los parámetros del juego.

```
void Start () {
    DefineRiddles();
    UpdateLevelData();
    ChangeRoomColor();
    SetRiddle();
    SetCorrectDoor();
    SetCluePosition();
}
```

Veamos cuál es la finalidad de cada función llamada desde Start().

```
public void DefineRiddles()
{
    riddles = new Riddle[,]
    {
        //Maths
        {
            new ImageRiddle(GameObject.Find("Pitagoras").GetComponent<Image>(),"Tales","Euclides","Pitágoras"),
            new TextRiddle("Número primo","147","171","139"),
            new TextRiddle("55 x 5","555","325","275"),
            new ImageRiddle(GameObject.Find("Square").GetComponent<Image>(),"4","8","16"),
            new TextRiddle("55-71","16","11","-16")
        },
        //History
        {
            new TextRiddle("II Guerra Mundial","1914-1918","1936-1939","1939-1945"),
            new ImageRiddle(GameObject.Find("Sea").GetComponent<Image>(),"Zeus","Apolo","Poseidón"),
            new ImageRiddle(GameObject.Find("Guillotine").GetComponent<Image>(),"1816","1492","1789"),
            new TextRiddle("Veni, vidi, vici","Octavio Augusto","Alejandro Magno","Julio César"),
            new AudioRiddle(GameObject.Find("Vivaldi").GetComponent<AudioSource>(),"Mozart","Beethoven","Vivaldi")
        },
        //Biology
        {
            new TextRiddle("Respiración","Estómago","Corazón","Pulmones"),
            new TextRiddle("Fotosíntesis","Raíces","Tallo","Hojas"),
            new AudioRiddle(GameObject.Find("HeartBeat").GetComponent<AudioSource>(),"Cerebro","Hígado","Corazón"),
            new AudioRiddle(GameObject.Find("Lion").GetComponent<AudioSource>(),"Herbívoro","Omnívoro","Carnívoro"),
            new ImageRiddle(GameObject.Find("Spider").GetComponent<Image>(),"Insecto","Molusco","Arácnido")
        }
    };
}
```

DefineRiddles, se encarga, como su nombre indica, de definir cuáles son los enigmas que el jugador se podría encontrar. En otras palabras, inicializa la variable *riddles*, que como ya se ha explicado, es una matriz que contiene objetos de la clase *Riddle*. Como podemos observar, para cada objeto construido no hay problema en llamar al constructor de su subclase, de manera que desde este momento establecemos el tipo de cada acertijo. Se aprecia que en cada fila de la matriz introducimos los enigmas de una de las asignaturas disponibles.

```
private void UpdateLevelData()
{
    //Actualizar el número del nivel
    if (PlayerPrefs.HasKey("level"))
    {
        level = PlayerPrefs.GetInt("level");
        level++;
    }
    else
        level = 1;
    if (level > lastLevel)
    {
        winningMenu.SetActive(true);
        PlayerPrefs.DeleteKey("level");
        PlayerPrefs.DeleteKey("topic");
        player.GetComponent<Rigidbody>().isKinematic = true;
    }
    else
    {
        PlayerPrefs.SetInt("level", level);
        levelText.text += level;
        //Recordar el tema elegido
        if (PlayerPrefs.HasKey("topic"))
        {
            topic = PlayerPrefs.GetInt("topic");
        }
        else
            topic = 1;
        topicText.text = GameData.topicNames[topic - 1];
    }
    chronometer = 120;
}
```

UpdateLevelData es una función que se encarga de tres tareas:

1. Actualizar los parámetros *level* y *topic* del nivel en el que nos encontramos. Para ello, aprovechamos la existencia de *PlayerPrefs*. Básicamente nos permite guardar variables en un fichero, para no perderlas cuando cambiemos de escena. Por ejemplo, podemos crear la variable de clave "level", de tipo entero en inicializarla a 1 cuando empezamos a jugar, ejecutando la instrucción *PlayerPrefs.SetInt("level", 1);* Lo que hacemos es comprobar si existen las variables que buscamos (en teoría siempre existen, pero siempre pueden aparecer errores inesperados). En caso de que existan las tomamos y las copiamos en las variables *level* y *topic*. A *level* le sumamos 1, ya que habremos alcanzado un nuevo nivel, mientras que *topic* lo dejamos como está. Si se ha producido algún error y las variables del fichero se han perdido, simplemente utilizamos 1 como valor por defecto para ambas variables.

2. Comprobar si hemos llegado al final del juego, es decir, si hemos conseguido superar todos los retos. Si es así, mostraremos el menú correspondiente para poder volver al menú principal.
3. Restablecer el tiempo límite de resolución de la prueba. Generalmente, utilizamos un tiempo de 120 s.

```
private void ChangeRoomColor()
{
    float red = GameData.lightColors[level - 1, 0];
    float green = GameData.lightColors[level - 1, 1];
    float blue = GameData.lightColors[level - 1, 2];
    wallMaterial.color = new Color(red, green, blue);
    clue.GetComponentInChildren<Text>().color = new Color(red+0.3f, green+0.3f, blue+0.3f);
}
```

ChangeRoomColor, es una función que se encarga de cambiar el color de la habitación, para que el juego no sea demasiado monótono. Su procedimiento es tomar una variable llamada *lightColors*, que es una lista de tuplas que representan colores en formato RGB y que se encuentra dentro de una clase llamada *GameData*. *GameData* es una clase estática que he utilizado para almacenar datos constantes que son utilizados por el controlador. Una vez tenemos el color que corresponde al nivel en el que nos encontramos, se lo asignamos al material de las paredes. También cambiamos el color del texto de las pistas para que, en el caso de que se trate de una pista de texto, no sea tan fácil de encontrar.

```
private void SetRiddle()
{
    //Colocar el acertijo según el nivel, el tipo y el tema
    myRiddle = riddles[topic - 1, level - 1];

    if (myRiddle is TextRiddle)
    {
        myRiddle = (TextRiddle)myRiddle;
        clue.GetComponentInChildren<Text>().text = ((TextRiddle)myRiddle).GetQuestion();
    }

    if (myRiddle is AudioRiddle)
    {
        ((AudioRiddle)myRiddle).GetAudio().loop = true;
        ((AudioRiddle)myRiddle).GetAudio().Play();
    }

    if (myRiddle is ImageRiddle)
    {
        ((ImageRiddle)myRiddle).GetImage().transform.localScale = new Vector3(1, 1, 1);
    }

    //Respuestas en las puertas
    for (int i = 0; i < 3; i++)
    {
        doors[i].GetComponentInChildren<Text>().text = myRiddle.GetAnswer(i);
    }
}
```

A diferencia de *DefineRiddles*, *SetRiddle* es una función que se centra exclusivamente en el enigma que debemos resolver en el nivel en el que nos encontramos. Básicamente, inicializamos la variable *myRiddle* con el acertijo de la matriz *riddles* correspondiente al nivel y al tema tratados.

Después realizamos algunas tareas en función del tipo de acertijo al que haremos frente:

- Si es de tipo texto: Asignamos el texto de *myRiddle* al *gameObject* que contiene la pista (*clue*). En este caso, *myRiddle* contiene un *string* con la pista textual, pero queremos ver físicamente esa pista en el juego. El objeto *clue* contiene un texto físico en el escenario, así que plasmamos la pista en ese objeto.
- Si es de tipo audio: Llamamos a la función *GetAudio* para tomar el archivo de audio que queremos reproducir y utilizamos Play para que se inicie dicha reproducción. Además, utilizamos el parámetro *loop*, para escuchar el sonido en bucle.
- Si es de tipo imagen: Las imágenes se encontrarán en el escenario con una escala de (0,0,0). Es decir, que no se verán hasta que cambiemos la escala. Por tanto, tenemos que cambiar la escala de la imagen que obtenemos al llamar a la función *GetImage*.

Una vez hecho esto, nos falta asignar a las inscripciones que se encuentran en las puertas, los textos contenidos en las posibles respuestas de *myRiddle*. Para ello, llamamos a la función *GetAnswer*:

```
private void SetCorrectDoor()
{
    Random.InitState((int)System.DateTime.Now.Ticks);
    correctDoor = Random.Range(1, 4);
    leftLimit = -7.5f + correctDoor * 3;
    rightLimit = leftLimit + 3;
    if (correctDoor == 1)
    {
        SwapPositions(doors[0], doors[2]);
    }
    else if (correctDoor == 2)
    {
        SwapPositions(doors[1], doors[2]);
    }
}
```

En la función *SetCorrectDoor*, se establece aleatoriamente cuál de las puertas será la que debemos atravesar para superar el reto, es decir, cuál contiene la inscripción correcta. Por la forma en la que están definidos los enigmas, la respuesta correcta siempre se encuentra, inicialmente, en la puerta 3. Solo teneos que generar un número aleatorio entre 1 y 3 e intercambiar la posición de la puerta correcta con la de la puerta 3, utilizando la función *SwapPositions*, que lleva a cabo el clásico algoritmo de intercambio.

```
private void SwapPositions(GameObject a, GameObject b) {
    Vector3 auxPosition;
    auxPosition = a.transform.position;
    a.transform.position = b.transform.position;
    b.transform.position = auxPosition;
}
```

Además, aprovechamos para establecer los límites espaciales que hacen que el jugador supere el nivel, los cuales dependen de cuál es la puerta buena.

```
private void SetCluePosition()
{
    if (myRiddle is ImageRiddle || myRiddle is TextRiddle)
    {
        Random.InitState((int)System.DateTime.Now.Ticks);
        int cluePosition = Random.Range(0, 4);
        clue.transform.position = new Vector3(
            GameData.cluePositions[cluePosition, 0],
            GameData.cluePositions[cluePosition, 1],
            GameData.cluePositions[cluePosition, 2]);
        clue.transform.rotation = Quaternion.Euler(
            GameData.clueRotations[cluePosition, 0],
            GameData.clueRotations[cluePosition, 1],
            GameData.clueRotations[cluePosition, 2]);
    }
}
```

Queremos que la posición de la pista para resolver el enigma sea aleatoria y para ello utilizamos la función *SetCluePosition*. Esto no tiene sentido en el caso de que la pista sea un sonido, pues no será visible. En otro caso, generamos un número aleatorio en el rango del número de posibles posiciones y accedemos a la variable *cluePositions y clueRotations*, que también pertenecen a la clase estática *GameData*. Le asignamos al objeto *clue* la posición y la rotación correspondientes (la rotación depende de la posición, intentando que la pista esté siempre pegada a la pared).

Ya hemos establecido todos los parámetros para comenzar a jugar, ahora simplemente debemos mantener el control de lo que sucede en el escenario. Para ello, en cada *frame* haremos una serie de comprobaciones, lo cual es posible gracias a la función *Update*.

```
// Update is called once per frame
void Update () {
    CheckLevelFinished();
    ReduceTimer();
}
```

En primer lugar, comprobamos si ha sucedido algo que produzca el final del nivel, ya sea para bien o para mal, llamando a la función CheckLevelFinished.

```
private void CheckLevelFinished()
{
    if (chronometer <= 0)
    {
        Loose();
    }
    if (player.transform.position.y < -2.5 && player.transform.position.y > -3.5)
    {
        if (player.transform.position.z > leftLimit && player.transform.position.z < rightLimit)
        {
            Win();
        }
        else
        {
            Loose();
        }
    }
}
```

Para empezar, si el valor del tiempo es menor o igual que 0, automáticamente hemos perdido. Después comprobamos si hemos atravesado el plano que se encuentra por debajo de la habitación y que indica que nos hemos lanzado por uno de los túneles para probar suerte. Si esto es así, solo hay que comprobar si la posición del jugador se encuentra dentro los límites establecidos anteriormente. Si lo está, hemos ganado, de lo contrario hemos perdido. Faltaría entonces definir las funciones Win() y Loose().

```
private void Win()
{
    if (myRiddle is AudioRiddle)
    {
        ((AudioRiddle)myRiddle).GetAudio().Stop();
    }
    SceneManager.LoadScene("3DoorsRoom", LoadSceneMode.Single);
}
```

Cuando ganamos (hemos acertado), lo primero que hacemos es, en el caso de que enigma fuese sonoro, detener la reproducción del audio, para evitar posibles problemas. Después, simplemente llamamos al SceneManager para volver a cargar la misma escena de nuevo (reiniciarla). Ahora, el parámetro “level” aumentará, así que el resto de los parámetros también cambiarán.

```
private void Loose()
{
    player.GetComponent<Rigidbody>().useGravity = false;
    room.SetActive(false);
    loosingMenu.SetActive(true);
    PlayerPrefs.DeleteKey("level");
    PlayerPrefs.DeleteKey("topic");
    if (myRiddle is AudioRiddle)
    {
        ((AudioRiddle)myRiddle).GetAudio().Stop();
    }
}
```

En caso de que fallemos, mostramos el menú de derrota. Después, borramos las variables "*level*" y "*topic*" que se encuentran en el fichero *PlayerPrefs*, ya que, de lo contrario, la próxima vez que juguemos no comenzaremos en el nivel 1, si no en el nivel que nos tocaba. También detenemos la reproducción del posible audio y, por último, hacemos que el cuerpo del jugador no se vea afectado por la gravedad, evitando que caiga al vacío eternamente.

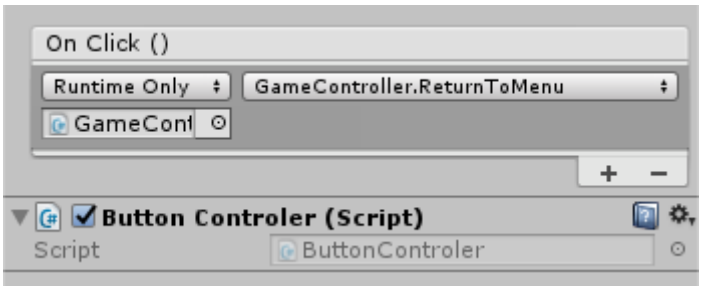
```
private void ReduceTimer()
{
    if (Time.timeSinceLevelLoad > 4 && chronometer > 0)
    {
        chronometer -= Time.deltaTime;
        int minutes = (int)chronometer / 60;
        int seconds = (int)chronometer % 60;

        timerText.text = minutes + ":" + seconds.ToString("00");

        if (chronometer <= 10 && timerText.color != Color.red)
            timerText.color = Color.red;
    }

}
```

ReduceTimer es una función que, como su nombre indica, se encarga de controlar en tiempo que nos queda para resolver el enigma. El crono comenzará a correr cuando hayan pasado 4 segundos desde que entramos en la habitación y lógicamente dejará de hacerlo cuando no nos quede tiempo. Reducimos la variable *float chronometer*, en una cantidad igual al tiempo que ha pasado desde el último frame. Después calculamos a cuántos minutos y segundos corresponde esa cantidad y lo plasmamos en la variable *timerText*, que como ya se ha explicado, es un objeto del escenario que contiene un texto. Además, si quedan menos de 10 segundos, el color del texto cambiará a rojo.

A continuación, explicaré otras funciones implementadas en la clase *GameController* que no son llamadas ni por Start ni por Update.

```
public void ReturnToMenu()
{
    player.GetComponent<Rigidbody>().useGravity = false;
    SceneManager.LoadScene("Main", LoadSceneMode.Single);
}
```

ReturnToMenu es una función que se ejecuta cuando queremos volver al menú principal. Existen tres casos en los que esto sucede: cuando queremos abandonar, cuando nos equivocamos de puerta o cuando superamos el reto. Recordemos que, en todos estos casos, existe un botón que debemos pulsar para confirmar nuestra decisión de retorno. Por tanto, lo que debemos hacer es llamar a la función cuando se pulsen dichos botones. No olvidemos que, para poder utilizar los botones, estos deben pertenecer a la clase *ButtonController*.




```
public void ExitMenuActivate()
{
    exit.SetActive(!exit.activeSelf);
}
```

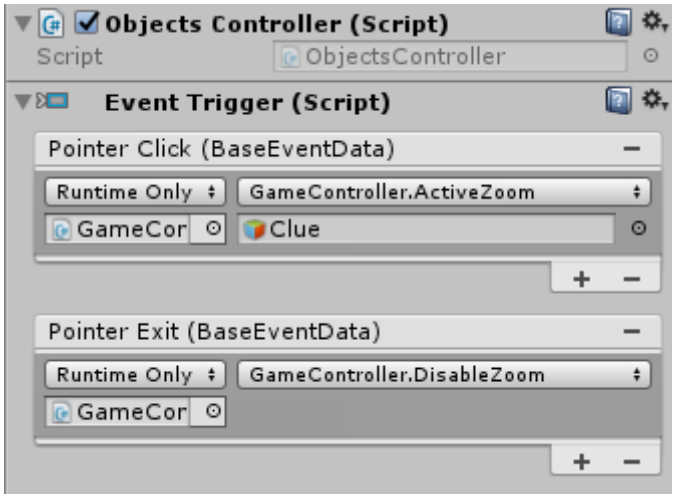
Continuamos con la función *ExitMenuActivate*, que muestra y esconde el menú de abandono, donde confirmamos si realmente queremos abandonar. Debemos llamarla cuando pulsamos el botón de escape, que se encuentra en la pared trasera de la habitación, utilizando la misma mecánica que en el caso anterior. También debemos utilizarla si se pulsa la opción "No" cuando este menú esté abierto, para cerrarlo.

```
public void ActiveZoom(GameObject objective)
{
    if (inverter.transform.localPosition.Equals(new Vector3(0, 1.8f, 0)))
    {
        float distance = Vector3.Distance(inverter.transform.position, objective.transform.position) - 2;
        inverter.transform.position += cam.transform.forward * distance;
        reticlePointer.enabled = false;
    }
}

public void DisableZoom()
{
    inverter.transform.localPosition = new Vector3(0, 1.8f, 0);
    reticlePointer.enabled = true;
}
```

Por último, contamos con las funciones *ActiveZoom* y *DisableZoom*. Mi intención era activar el zoom automáticamente al fijar la mirada sobre una pista o inscripción. El método tradicional para efectuar un zoom es disminuir el parámetro *field of view* de la cámara. Sin embargo, esto no funciona en realidad virtual, por lo que tuve que recurrir a otro método. Opté por, simplemente, acercar la cámara hacia el objetivo, para simular el zoom.

Así pues, la función *ActiveZoom* recibe como parámetro el objeto al que queremos acercar la cámara. Se calcula la distancia que esta debe desplazarse, que es igual a la distancia entre el jugador y el objetivo, menos 2. Después, la cámara se desplaza esa distancia en la dirección en la que está mirando. También se hace invisible al puntero, para evitar que dificulte la visión de aquello que queremos observar con detalle. *ActiveZoom* se ejecuta cuando "pulsamos" (miramos fijamente) aquellos objetos descritos anteriormente. Por tanto, estos objetos deben pertenecer a la clase *ObjectController* y tener un *EventTrigger* que llame a esta función al realizar un clic.

Mientras que *DisableZoom*, simplemente devuelve la cámara a su posición original y hace el puntero nuevamente visible. Se ejecuta cuando dejamos de mirar al objeto.



5.3.9 La clase estática GameData

Esta es una clase creada con el objetivo de almacenar parámetros constantes, tales como las posibles posiciones de las pistas, los colores de las habitaciones o los nombres de las asignaturas. No tiene mucho más que explicar, así que simplemente adjuntaré el código.

```
public static class GameData {


    public const int MATHS = 1;
    public const int HISTORY = 2;
    public const int BIOLOGY = 3;

    public const int TEXT_TYPE = 1;
    public const int IMAGE_TYPE = 2;
    public const int SOUND_TYPE = 3;

    public static AudioSource heartBeat = GameObject.Find("HeartBeat").GetComponent<AudioSource>();
    public static AudioSource lion = GameObject.Find("Lion").GetComponent<AudioSource>();



    public static string[] topicNames = new string[] { "MATEMÁTICAS", "HISTORIA", "BIOLOGÍA" };


    public static float[,] lightColors = new float[,] {
        { 1, 0.8f, 0.1f },
        { 0, 0.5f, 1 },
        { 0.9f, 0.3f, 0.5f },
        { 0.4f, 0.8f, 0 },
        { 0, 0, 1 },
        { 0.1f, 0.3f, 0 },
        { 1, 1, 0 },
        { 1, 0.5f, 0 },
        { 1, 0, 0 },
        { 1, 0, 0.5f },
        { 1, 0, 1},
        { 0.5f, 0, 1}
    };
```

```
    public static float[,] cluePositions = new float[,]
    {
        { 4.99f, 4.2f, -3.5f },
        { 3.5f, 0.2f, -5 },
        { 4, 4, 5 },
        { -4.3f, 1, 5 },
        { -3.54f, 3.85f, -4.99f },
        { 1, 4.99f, 4 },
        { -2, 4.99f, -3 },
        { 1, 0.5f, 0 },
        { 1, 0, 0 },
        { 1, 0, 0.5f },
        { 1, 0, 1},
        { 0.5f, 0, 1}
    };

    public static float[,] clueRotations = new float[,]
    {
        { 0, 90, 0 },
        { 0, 180, 0 },
        { 0, 0, -45 },
        { 0, 0, 90 },
        { 0, 180, -35 },
        { -90, 0, 0 },
        { -90, 180, 90 },
        { 1, 0.5f, 0 },
        { 1, 0, 0 },
        { 1, 0, 0.5f },
        { 1, 0, 1},
        { 0.5f, 0, 1}
    };

}
```

5.3.10 El tutorial

Como se ha explicado anteriormente, el tutorial del juego consiste únicamente en un conjunto de paneles que nos muestran, mediante texto e imágenes, el funcionamiento de la aplicación. Estos paneles se encuentran dentro de un canvas, junto a los botones de avanzar y retroceder, que nos ayudan a navegar entre ellos. Hemos creado la clase Tutorial, que contiene varias funciones que son de utilidad en esta escena.

```
public class Tutorial: MonoBehaviour {

    private int currentSlide;
    public int maxSlide;
    public GameObject[] slides;
    public GameObject player;
    public GameObject panel;
    public Text nextButton;
    public Text backButton;
    // Use this for initialization
    void Start () {
        currentSlide = 0;
    }
```

La clase contiene las siguientes variables:

- *currentSlide:* Un número entero que indica el índice del panel que se muestra actualmente en la escena.
- *maxSlide:* Es el número de paneles que hay en total.
- *Slides:* Es un array de objetos que contiene los paneles en sí.
- Player: Objeto correspondiente al jugador.
- Panel: Es el panel oscuro que se encuentra detrás de los otros paneles.
- *nextButton:* Es el texto contenido en el botón de avanzar, que nos interesa porque puede variar según la diapositiva en la que nos encontremos.
- *backButton:* Es el texto contenido en el botón de retroceder, que al igual que el anterior, puede variar.

Observamos que lo único que hacemos al instanciar la clase es inicializar la variable *currentSlide* a 0.

```
public void Next()
{
    if (currentSlide >= maxSlide)
    {
        SceneManager.LoadScene("Main", LoadSceneMode.Single);
    }
    else {
        StartCoroutine(TurnRight());
    }
}

public void Back()
{
    if (currentSlide <= 0)
    {
        SceneManager.LoadScene("Main", LoadSceneMode.Single);
    }
    else
    {
        StartCoroutine(TurnLeft());
    }
}
```

Tenemos también dos funciones que son respectivamente, aquellas a las que llamaremos cuando pulsemos los botones de avanzar y retroceder. Next comprueba si nos encontramos en la última diapositiva y si es así, nos devuelve al menú principal. De lo contrario, llama a la corrutina *TurnRight*. Análogamente, Back comprueba si nos encontramos en la primera diapositiva y en ese caso nos devuelve al menú principal. En otro caso, llama a la corrutina *TurnLeft*. Expliquemos pues, el funcionamiento de estas corrutinas.

```
    private IEnumerator TurnRight()
    {
        slides[currentSlide].SetActive(false);
        panel.SetActive(false);
        currentSlide++;
        for (int i=0; i<90; i++)
        {

            player.transform.Rotate(0, 1, 0);
            yield return new WaitForSeconds(0.01f);

        }
        panel.SetActive(true);
        slides[currentSlide].SetActive(true);
        if (currentSlide == maxSlide)
            nextButton.text = "TERMINAR";
        else if (currentSlide == 1)
            backButton.text = "ATRÁS";

    }

    private IEnumerator TurnLeft()
    {
        slides[currentSlide].SetActive(false);
        panel.SetActive(false);
        currentSlide--;
        for (int i = 0; i < 90; i++)
        {

            player.transform.Rotate(0, -1, 0);
            yield return new WaitForSeconds(0.01f);

        }
        panel.SetActive(true);
        slides[currentSlide].SetActive(true);
        if (currentSlide <= 0)
            backButton.text = "SALIR";
        else if (currentSlide == maxSlide - 1)
            nextButton.text = "SIGUIENTE";

    }
}
```

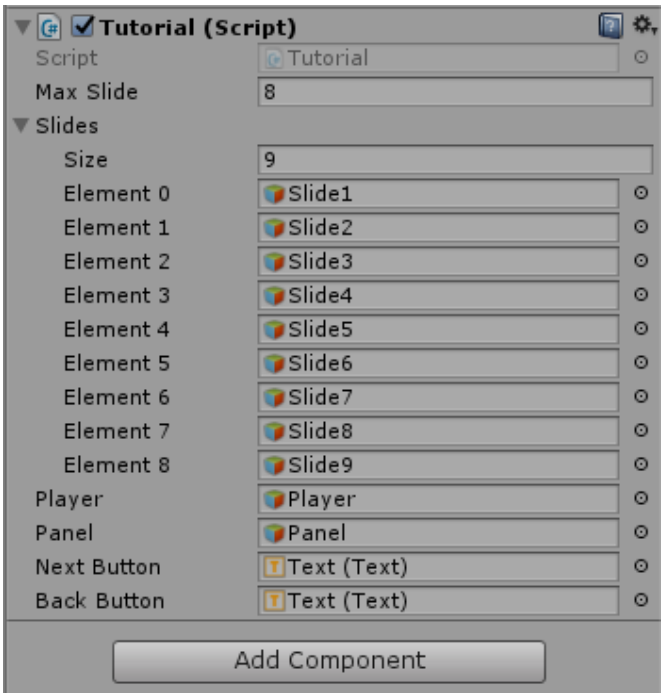
El funcionamiento se resume de la siguiente forma:

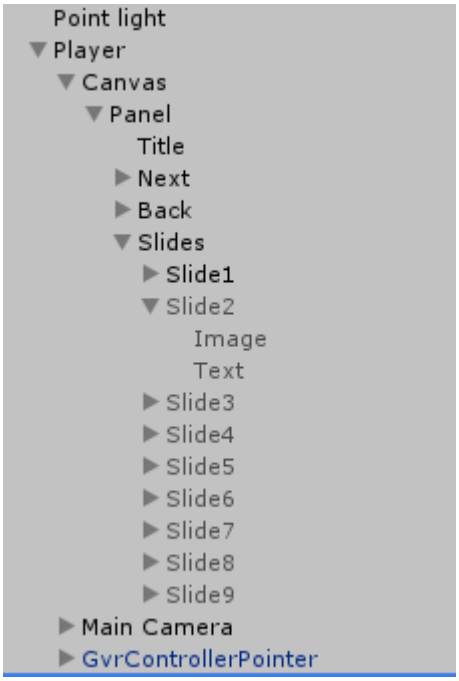
- Si queremos avanzar, escondemos el panel actual y actualizamos el índice sumando uno a *currentSlide*. Rotamos 90 grados a la derecha, en un tiempo de 0.9 segundos. Cabe destacar que el canvas se encuentra, en la jerarquía como hijo del jugador, por lo que al girar nosotros, también lo hace el canvas (en torno a nuestra cabeza). Volvemos a mostrar el panel actual, que será el siguiente al

que hemos escondido y, si es necesario, cambiamos el texto de los botones como se muestra.

- Si queremos retroceder, también escondemos el panel, pero esta vez, restamos uno a la variable *currentSlide*. Rotamos a la izquierda y al igual que antes, volvemos a mostrar el panel (será el anterior al que hemos escondido) y a actualizamos los textos en caso de ser necesario.

Con esta clase creada, lo que hacemos es colocarla como componente del canvas (entonces el canvas es de la clase Tutorial). Desde el interfaz de Unity, podemos asignar las variables que hemos declarado como públicas, incluyendo los objetos y textos.





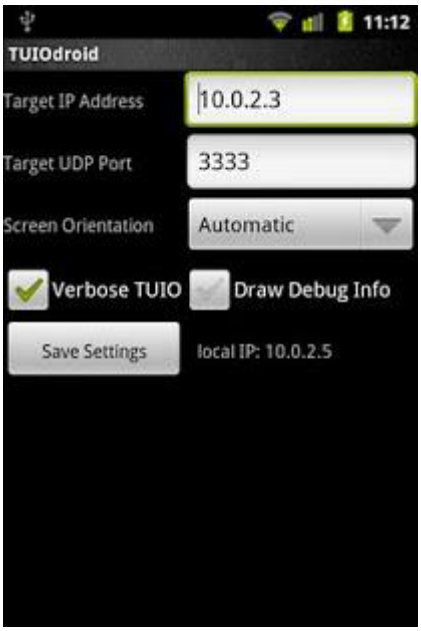
Como se aprecia, tenemos un total de 9 diapositivas (desde la 0 hasta la 8). Muestro también la jerarquía de la escena para comprender la estructura:

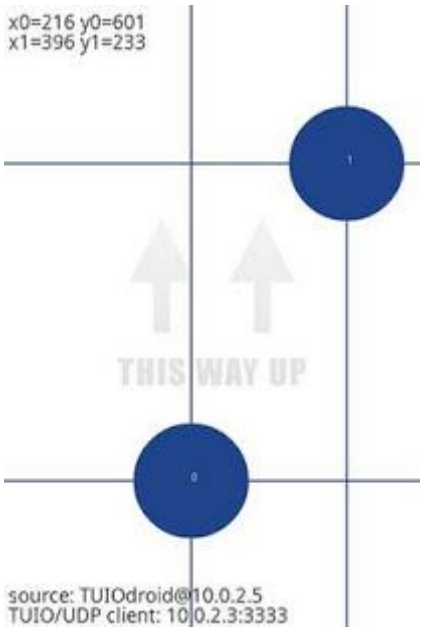
- El jugador es el padre del interfaz y de la cámara. Cuando movemos la cabeza es la cámara la que gira, pero cuando cambiamos de panel es el jugador al completo quien lo hace.
- El canvas contiene un título, los botones (Next y Back) y el objeto *Slides*.
- *Slides* contiene las diferentes diapositivas (Slide1, Slide2…), cada una de las cuales contiene un texto y una imagen.

6. Problemas encontrados

A continuación, enumero y explico los problemas que hemos encontrado durante la elaboración de este proyecto.

En primer lugar, nos encontramos con la imposibilidad de tocar la pantalla cuando estamos ejecutando la aplicación con las gafas puestas. Ya hemos explicado la solución final, que consiste en activar la interacción con los objetos cuando los miramos fijamente. Pero antes de llegar a esta solución, planteamos otra que no resultó. Descubrimos que existía un sistema mediante el cual se podía controlar la pantalla de nuestro dispositivo desde otro dispositivo remoto. Para ello, debíamos conectar ambos dispositivos a la misma red y descargarnos una aplicación llamada Tuio Droid en el dispositivo remoto. Además, antes de montar la aplicación, teníamos que incluir elementos de una librería especial llamada Tuio Input en el proyecto de Unity. Después solo teníamos que ejecutar Tuio Droid en el dispositivo remoto e introducir los datos de IP y puerto de la aplicación, para que todo funcionase.





Al final, esta opción fue descartada por varias razones. Para empezar, es un problema el hecho de necesitar dos dispositivos para poder jugar. Desde luego, si comercializáramos esta aplicación, esto nos limitaría bastante. Además, no solo necesitas tener dos dispositivos, si no que también debes descargarte otra aplicación en introducir datos como la IP del dispositivo, algo de lo cual no todo el mundo es capaz. Otro problema de este sistema es que la aplicación Tuio Droid entra en el modo de configuración en cuanto mueves bruscamente el teléfono. Esto es engorroso porque nos movemos mucho cuando estamos jugando y ni siquiera nos damos cuenta al instante cuando esto pasa. Por todas estas razones, decidimos rechazar esta opción.

Otro problema encontrado fue la creación del modo invertido. Es sabido que, al utilizar este modo, la inversión solo se produce en la rotación sobre el eje vertical. Pero la intención que teníamos en un principio era la inversión total de la rotación. Lo cierto es que al final lo descartamos porque no conseguimos que funcionase correctamente.

También hubo algunos obstáculos relacionados con la portabilidad entre el interfaz de Unity y el teléfono. Por ejemplo, no podíamos usar métodos que abriesen archivos de imagen o audio, puesto que, al montar la aplicación, solo montamos lo que tenemos en la escena y no todas las carpetas que se encuentran en el proyecto. La solución es sencilla, solo debemos introducir estos archivos en la escena y tratarlos como *game objects*.

7. Pruebas y test de usabilidad

En la elaboración de este proyecto, han tenido lugar varias pruebas que nos han ayudado a conseguir el resultado final.

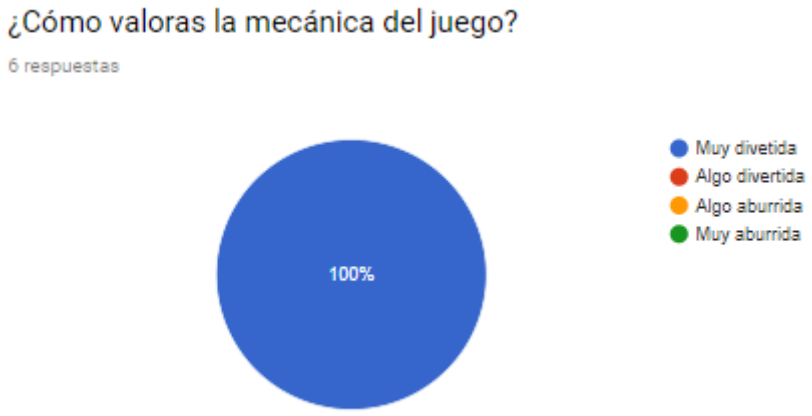
Antes de iniciar el diseño de la aplicación en sí, estuvimos probando algunos juegos que construimos simplemente para comprobar el funcionamiento de la realidad virtual. Uno de ellos consistía en un túnel por el cual, el jugador avanzaba automáticamente (aun no sabíamos cómo controlar el desplazamiento). Realmente era una idea de posible proyecto final que terminamos descartando. Con esta aplicación probamos también el modo invertido por primera vez.

También intentamos realizar modificaciones en la aplicación que he comentado en la introducción, la de eliminar formas geométricas, pero no son de gran trascendencia.

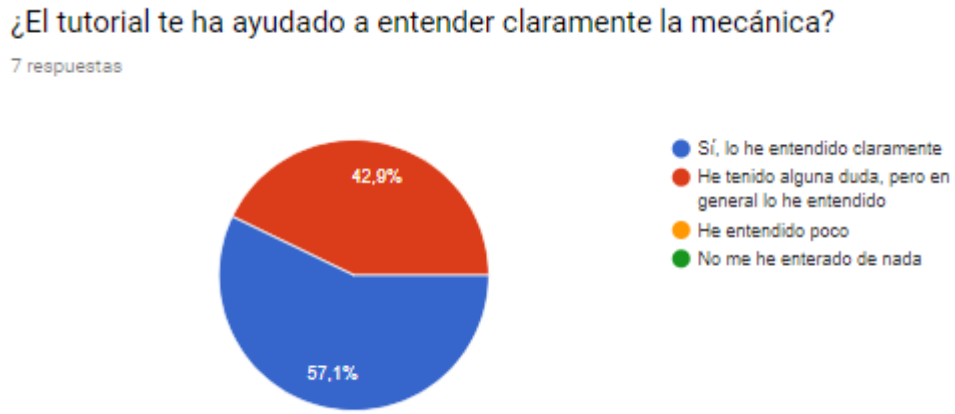
Una de las pruebas más trascendentes la realizamos con una versión casi definitiva de la aplicación. La prueba consistía en reunir a unos cuantos amigos y familiares para que realizar una demostración de la aplicación:

- En primer lugar, realicé una explicación muy sencilla sobre cómo interactuar con los botones y poco más, ya que la intención era que los usuarios aprendieran por sí mismos el funcionamiento y la mecánica del juego a través del tutorial que se incluye.
- Después, por turnos, cada persona probaba la aplicación, siguiendo mis instrucciones:
    - Primero, seleccionaba el tutorial y lo leía por completo, para aprender a jugar.
    - Después, jugaba al modo normal, eligiendo la asignatura que la persona deseara, hasta caer eliminado.
    - Por último, jugaba al modo invertido, también hasta eliminarse.
- Finalmente, cada usuario rellenaba un test de usabilidad anónimo, con el objetivo de encontrar las virtudes y defectos de la aplicación y poder mejorarla.

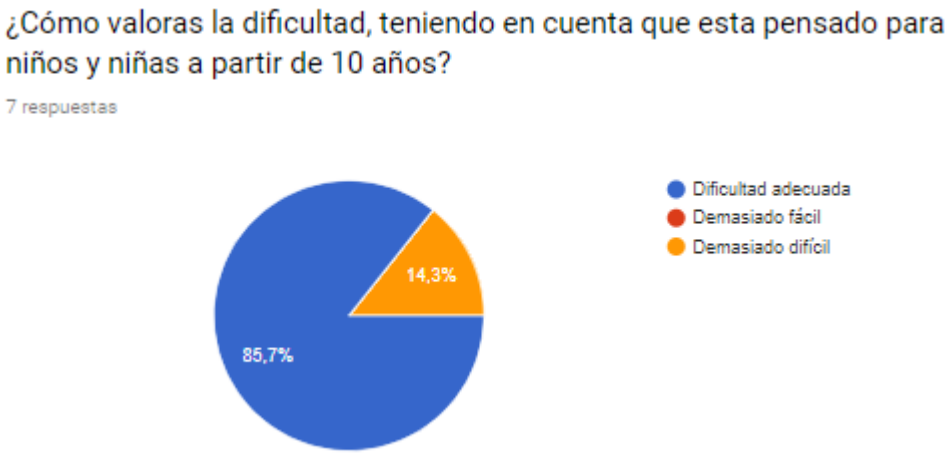
Ahora mostraré las preguntas que aparecían en dicho test para analizarlas una por una.



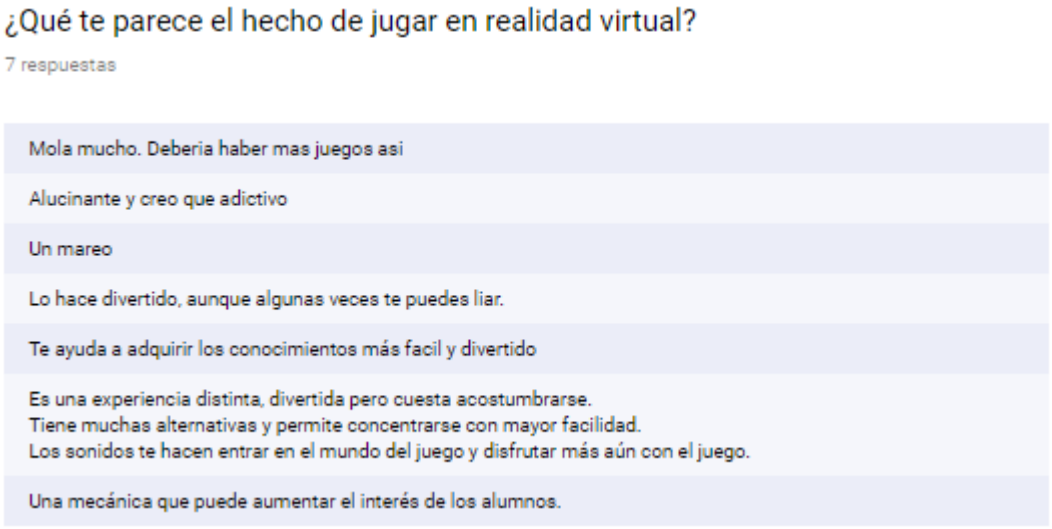
La primera pregunta está relacionada con la mecánica del juego. Se puede decir, por unanimidad, que al público que probó la aplicación le parece divertida.



La segunda pregunta hace referencia al tutorial. En general, se ha valorado positivamente. Algo más de la mitad de los encuestados piensan que el tutorial es suficiente para entender correctamente la mecánica del juego, mientras que el resto piensan que puede generar alguna duda, pero en general lo han entendido. En este sentido, quizá podamos mejorar el tutorial, aunque realmente la intención es que algunas cosas no se desvelen para poder sorprender al usuario durante la experiencia de juego.



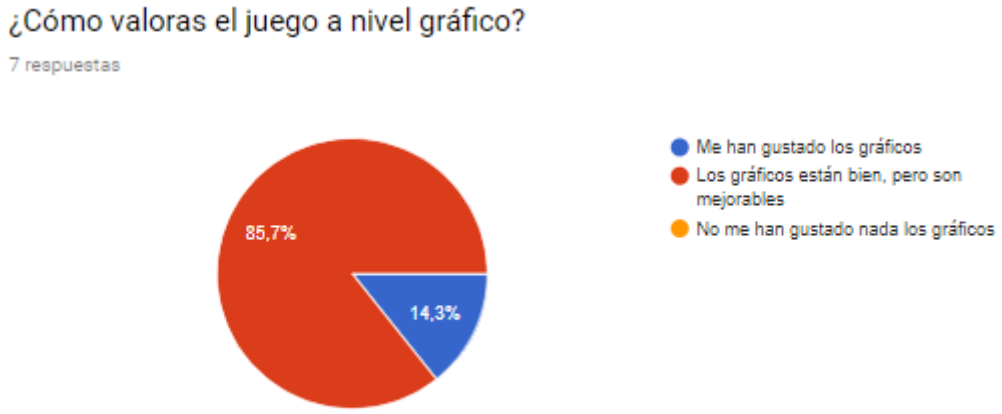
En tercer lugar, hemos querido saber si la dificultad era la adecuada. Más del 80% de los encuestados piensa que la dificultad es adecuada, mientras que el resto piensa que es demasiado difícil. Con estos resultados podemos decir que tal vez la dificultad dependa de la edad de los chavales. En general, conforme avanzamos en un reto, los enigmas se van complicando, pero en base a los resultados no considero que se deban realizar cambios en este aspecto.



También he querido vislumbrar opiniones concretas sobre el hecho de jugar utilizando la realidad virtual. De entre las frases que se observan, quiero destacar algunas:

- “Un mareo”: Es cierto, las cosas claras, el juego marea. Es uno de sus puntos flacos, aunque como se puede ver en otras opiniones, es cuestión de acostumbrarse.
- “Una mecánica que puede aumentar el interés de los alumnos.”: Es uno de los objetivos del juego, ya que últimamente es complicado que los alumnos mantengan la concentración o el interés en las clases.

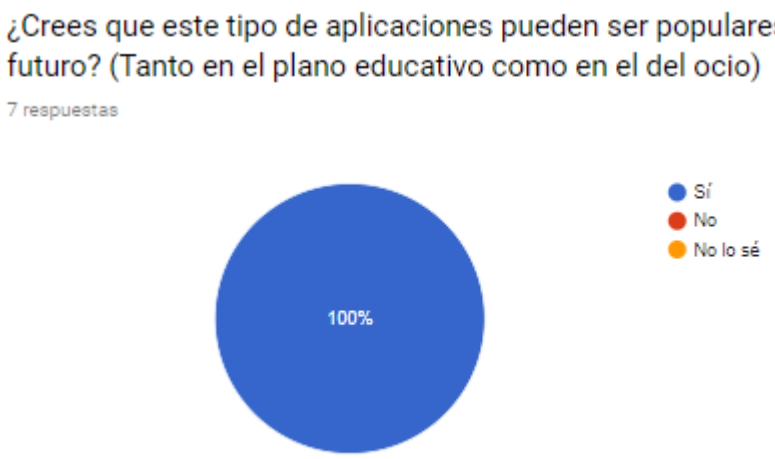
En cuanto a los controles, como era de esperar, a la mayoría de los usuarios le ha costado adaptarse. Como ya he mencionado anteriormente, es una aplicación diferente y requiere de tiempo para dominarla, pero hemos conseguido que ningún usuario opine que es demasiado incómodo.



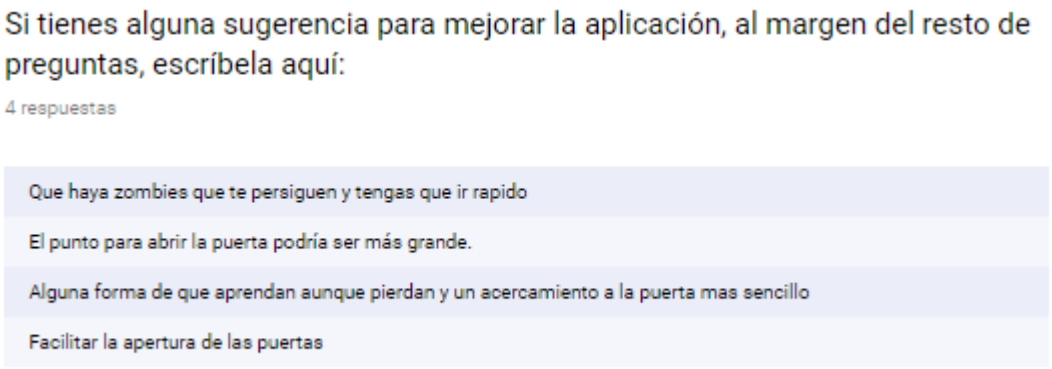
Claramente, se puede afirmar que los gráficos del juego son mejorables. Es cierto que no es el aspecto en el cual nos hemos centrado, ya que hemos hecho mayor hincapié en el apartado lógico. Será un apartado a mejorar en futuras versiones de la aplicación.

El modo invertido es uno de los aspectos que ha tenido resultados más negativos, pues, aunque es cierto que muchas personas lo valoran como divertido, no es este el objetivo de dicho modo. Además, un porcentaje relativamente elevado no le ven ninguna utilidad y tan solo un 14% piensa que es realmente útil. Por tanto, en la próxima versión de la aplicación, tendremos que valorar como podemos mejorar este modo de juego y quizá nos planteemos su eliminación.

La siguiente pregunta hace referencia al que es el objetivo principal de la aplicación. En general, hago un balance positivo de los resultados, pues todos los usuarios piensan que la aplicación tiene al menos algo de utilidad en el plano educativo. La intención no es que esta aplicación tenga una utilización muy frecuente en la enseñanza, sino que se utilice esporádicamente, por lo que quizá se haya sobrevalorado su utilidad, pero aun así podemos valorar los resultados como positivos.



En cuanto a la popularidad que puede alcanzar este tipo de aplicaciones, la totalidad de los encuestados piensan que esta se alcanzará en un futuro. No solo en el plano educativo sino también en el del ocio.



Por último, quise añadir una pregunta para que los usuarios añadiesen sugerencias libremente. En ella, es interesante que varios usuarios opinaban que la apertura de la puerta era difícil. Según me explicaron, es complicado atinar para mirar el pomo de la puerta, que es lo que permite abrirla. Por ello, en futuras versiones plantearemos soluciones para este problema, ya sea utilizar un pomo más grande o bien cambiar el sistema.

8. Resultado final y posibles mejoras

Finalmente, tras realizar varias pruebas en teléfonos personales, comprobamos un correcto funcionamiento de cada apartado explicado en la sección de la implementación. Al abrir la aplicación, tenemos que colocar nuestro dispositivo en las gafas de realidad virtual. Veremos el menú principal con todas las opciones ya explicadas. Lo recomendable es comenzar echando un vistazo al tutorial, no solo para comprender el funcionamiento si no también para que nuestra visión se acostumbre antes de dar paso a la acción.

Ahora mismo, se trata de una versión muy básica de la aplicación en cuanto a la variedad que ofrece. Por el momento, podemos trabajar tres asignaturas: matemáticas, historia y biología. Cada asignatura cuenta con cinco acertijos para llegar al final. La intención es ampliar tanto la cantidad de asignaturas disponibles como el número de niveles explorables en cada una de ellas.

Existen algunas mejoras claras para esta aplicación y otras que dependen de la situación y de las preferencias del usuario.

En cuanto a la jugabilidad de este proyecto, esta es aún muy básica, por lo que se me han ocurrido varias mejoras en este aspecto:

- **La posibilidad de que el jugador introduzca sus propios acertijos:** En este sentido, sería importante aplicar la mejora comentada antes a cerca de guardar los archivos en el servidor. Los jugadores podrían enfrentarse a retos que proponga la comunidad. Para ello, también necesitaríamos una base de datos en la que guardar los nuevos enigmas, además de un sistema de aprobación.
- **Introducir métodos más complejos para resolver los acertijos:** El juego sería más divertido si, en lugar de únicamente tener que buscar la pista por la habitación, hubiera que hacer algo más complejo para poder conocerla. Por ejemplo, buscar llaves ocultas, las cuales abren las puertas o cofres que contienen la pista definitiva; también podemos utilizar una habitación con mas recovecos que dificulten el hallazgo de los elementos.
- **La posibilidad de vislumbrar parámetros,** como el tiempo que tardamos en superar el reto, estableciendo récords y demás.
- **Un sistema de configuración,** que nos permita establecer niveles de dificultad de los retos, el tiempo que tenemos para superarlos, diferentes escenarios, etc.

Existen también algunos cambios que podrían mejorar el juego de cara a la usabilidad:

- Es sabido que utilizar el teléfono móvil con las gafas de realidad virtual puestas durante un período de tiempo muy prolongado, puede perjudicar nuestra visión. Podría agregarse un sistema que cronometre el tiempo que llevamos con

la aplicación abierta y, en caso de que este tiempo supere, por ejemplo, la media hora, apareciera un mensaje de aviso.

- Además, podríamos implantar un modo de juego para personas con discapacidad auditiva, evitando aquellos enigmas en los que sea necesario escuchar para poder resolverlos.

9. Conclusiones

Teniendo en cuenta la naturaleza del proyecto, considero que su mayor valor de cara al público radica en la posibilidad que ofrece de ejercitar la mente y al mismo tiempo entretenernos de una manera moderna y novedosa. Realmente, se trata de una apuesta de cara al futuro, ya que podemos decir que, actualmente, la mayoría de las personas no están familiarizadas con la realidad virtual. Si visitas la vivienda de algún amigo, lo más probable es que no te encuentres con unas gafas VR. Pero lo mismo podríamos haber dicho hace no tantos años del teléfono móvil y ahora solo hay que abrir los ojos para ver lo que sucede, como ocurre con toda novedad tecnológica. Lo cierto es que cada vez es mayor el porcentaje de teléfonos o tabletas fabricados compatibles con la realidad virtual. Con la modernización de la enseñanza y la cada vez más fuerte implantación de la tecnología en la misma, no sería tan descabellado pensar en la utilización de este tipo de aplicaciones en las escuelas, en un futuro no muy lejano.

Encontramos también algunos contras y es que no se trata de una aplicación que puedas utilizar durante un intervalo de tiempo muy prolongado. El motivo es que, al cabo de unos minutos, nuestra visión comienza a sentir cansancio y debemos descansar. El peligro es hacer caso omiso de esta sensación de cansancio y terminar dañando nuestro cuerpo, más teniendo en cuenta que el público objetivo son niños de a partir de diez años. Por eso, quiero recomendar el uso moderado y controlado por los padres o tutores de este producto.

En cuanto a la aportación personal que me llevo por trabajar en este proyecto, puedo enumerar varios aspectos. En primer lugar, esta claro que el aprendizaje ha sido de gran valor. Me he formado en un campo que tal vez actualmente no tenga un gran impacto, pero es posible que de cara al futuro me ofrezca pequeñas o grandes oportunidades. Además, he descubierto un campo que me gusta, no he sufrido por tener que pasar horas y horas probando cosas nuevas, sino que lo disfrutaba. También me ha hecho a crecer en cuanto a madurez en el trabajo, puesto que he tenido gran libertad en cuanto a la forma de llevar a cabo cada tarea y eso me ha ayudado a tener una visión mayor de los métodos que funcionan mejor y peor.

Dicho esto, en este momento en el que finalizo los estudios de este grado, agradezco la oportunidad que brinda la Universidad Pública de Navarra respecto a la posibilidad de adquirir formación de gran calidad y de manera asequible para la mayoría de las personas. Agradezco también la labor de los profesores, en especial la de Oscar Ardaiz, sin el cual no habría sido posible la elaboración de este proyecto.

10. Bibliografía

Para poder realizar este proyecto, en muchos casos he tenido que recurrir a la búsqueda de información. Generalmente se ha tratado de situaciones en las que no he sido conocedor de elementos del lenguaje C Sharp. En internet encontramos varios sitios en los que se recopilan guías de programación de este lenguaje, especialmente orientado a Unity, además de guía de utilización del motor:

https://unity3d.com

https://answers.unity.com/

El primero de ellos es una guía en la que he podido aprender conceptos básicos y se muestran ejemplos de cómo aplicarlos. El segundo se trata, básicamente de un foro en el que he podido encontrar soluciones a problemas más complejos.

Además de eso, para poder comprender el funcionamiento de los elementos de realidad virtual, concretamente la librería Google VR, he visitado el siguiente sitio:

https://developers.google.com/vr/develop/unity/get-started-android