Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos.
UNIVERSIDAD DE CANTABRIA

VIRTUALIZACIÓN DEL PROCESO DE CREACIÓN DEL HORMIGÓN

TRABAJO FINAL DE MASTER

Trabajo realizado por:

Alain Villanueva Merino

Dirigido:

Miguel Cuartas Hernández

Valentín Arroyo Fernández

Titulación:

Máster Universitario en Ingeniería de Caminos, Canales y Puertos

Santander, septiembre de 2018

ÍNDICE

1. RESUMEN

1.1. RESUMEN

La realidad virtual es un entorno artificial creado mediante un software informático. El objetivo principal es que el usuario encuentre y acepte dicho entorno como si de uno real se tratase. Para conseguirlo, se usa un ordenador o consola que intenta transmitir una experiencia lo más real posible mediante dos de los cinco sentidos, la vista y el oído.

Al igual que otras muchas herramientas relacionadas con la informática, la realidad virtual está teniendo un desarrollo muy veloz. Por consiguiente, se pueden encontrar diferentes variedades tanto de dispositivos de realidad virtual, como de programas para desarrollar los entornos artificiales.

Entre los diferentes programas, para la realización de esta aplicación se ha optado por el Unity 3D. Las razones de mayor peso son la facilidad para importar elementos desde diferentes programas, la interfaz intuitiva de la que dispone y la presencia de la Unity Store.

En relación con los diferentes usos que se le pueden dar a la realidad virtual hoy en día, podemos encontrar dos grandes grupos. El primero sería la simulación de diferentes ambientes para el entrenamiento, trabajo o la educación, y el segundo el desarrollo de ambientes ficticios para la creación de juegos o historias interactivas.

El presente trabajo estaría incluido en el primer grupo, más concretamente en el área de la educación; ya que se ha diseñado un laboratorio virtual en el cual se puede simular la creación de hormigón interactuando con los diferentes objetos añadidos.

El programa realizado, por un lado, aprovecha la velocidad de computación de un ordenador. Porque al igual que en la realidad, el programa nos ofrecerá la posibilidad de utilizar diferentes cantidades de los elementos necesarios para la realización del hormigón; y dependiendo de que cantidades se usen, el ordenador realizará unos cálculos u otros, con los que se obtendrán diferentes resultados. Además, al tratarse de un entorno virtual, se podrá realizar toda la cantidad de combinaciones diferentes que se deseen, obteniendo el resultado de cada combinación al momento.

Por otro lado, la realidad virtual ofrece la posibilidad de interactuar directamente con los componentes, por lo que generará un mayor grado de atracción en el usuario, y, en consecuencia, una mayor retención de la información y un mejor aprendizaje.

Para finalizar, mencionar la importancia de la parte de la programación en el trabajo. Ha sido necesaria la creación de diferentes *Scripts* en el lenguaje de programación *C#* para hacer posible la interacción entre el usuario y los objetos y la realización de los cálculos necesarios.

1.2. ABSTRACT

The Virtual Reality is an environment created by a software simulation. The main objective is that the user feels and accepts the environment like a real one. In order to achieve it, the computer or the console tries to transmit the most realistic experience by two of the five senses, sight and hearing.

As many other tools related to computer science, virtual reality is developing fast. Therefore, you can find different varieties of virtual reality devices, as well as programs to develop artificial environments.

Among the different programs, for the realization of this application we have opted for Unity 3D. The most important reasons are the facilities to import elements from different programs, the intuitive interface and the presence of the Unity Store.

In relation to the different uses that today can be given to virtual reality, we can find two large groups. The first would be the simulation of different environments for training, work or education, and the second the development of fictitious environments for the creation of games or interactive stories.

The present work would be included in the first group, more specifically in the area of education. There has been designed a virtual laboratory where the creation of concrete can be simulated by interacting with the different objects.

On the one hand, program takes advantage of the computing speed of a computer. As in reality, the program offers us the possibility of using different quantities of the elements necessary for the realization of concrete and depending on them, the computer performs different calculations, which give different results. In addition, all the different combinations can be made obtaining the results at the moment.

On the other hand, virtual reality offers the possibility of interacting directly with the components, which will generate a greater degree of attraction in the user, and, consequently, a greater retention of information and a better learning.

Finally, mention the importance of the part of programming in the work. It has been necessary to create different scripts in the programming language C# to

make possible the interaction between the user and the objects as well as the realization of the necessary calculations.

2. INTRODUCCIÓN

2.1. CONTEXTO

La realidad virtual, tal y como se conoce hoy, nace de unas ideas que pueden datarse a principios del siglo XIX. Para ser exactos, en 1838 se creaba el primer estereoscopio. Este aparato contenía dos espejos que proyectaban una misma imagen a la que se la conseguía dar una cierta profundidad.

El término “Realidad Virtual” se usa por primera vez a mediados de los ochenta, cuando Jaron Lanier empieza a desarrollar un mecanismo formado por gafas y guantes en busca de hacer lo más real posible dicha experiencia.

Fueron los años setenta y los ochenta los más estimulantes para el desarrollo de esta tecnología. La razón fue que se desarrollaron grandes avances tanto en la tecnología óptica como en la de aparatos táctiles, ambos con una gran relación con la realidad virtual. Como ejemplo, a mediados de los ochenta, en el centro de investigación de la NASA se desarrolla un aparato con un casco y unos guantes que permiten una interacción con diferentes aparatos táctiles llamado Virtual Interface Environment Workstation.

En el presente, la Realidad Virtual está en su momento más importante y el número de empresas que desarrollan productos relacionados con la realidad virtual es cada vez mayor. Esto se debe a los diferentes usos que tiene. Entre ellos se pueden destacar los siguientes:

- Videojuegos: es el área con mayor potencial de la realidad virtual. Gracias a esta mejora, la gente puede sumergirse totalmente en el videojuego, haciendo una experiencia mucho más real e inmersiva.

- Entretenimiento: el ejemplo más claro es el uso en el cine. Cada vez se está volviendo más común la aparición de películas con versión en 3D.

- Educación y entrenamiento: esta área es de las más amplias, ya que abarca desde un piloto de avión entrenándose mediante la realidad virtual, a un estudiante de medicina simulando una operación.

- Psicología: como con la realidad virtual se puede simular cualquier situación, se pueden recrear diferentes situaciones de fobias para ayudar a la gente a eliminarlas, sabiendo que es un ambiente ficticio y que el riesgo es nulo.

- Ingeniería: muchos de los prototipos que crean las empresas pueden recrearse en la realidad virtual para obtener una primera impresión visual, y de esta manera evitar el gasto de tiempo y materiales.

- Arquitectura: mediante la realidad virtual se puede crear cualquier construcción a cualquier escala, incluso pudiendo simular el interior de dicha construcción.

2.2. FUNCIONAMIENTO DE LOS DISPOSITIVOS DE REALIDAD VIRTUAL

El objetivo principal de estos dispositivos es el de crear una realidad virtual, es decir, el de crear un nuevo ambiente virtual, pero lo más parecido posible a la realidad. Para ello, estos dispositivos actúan tanto como un aparato especializado de entrada de información, como un aparato especializado de salida de información.

Para funcionar como aparato de entrada, los dispositivos utilizan diferentes sensores que permiten a la aplicación consultar la orientación y la posición de la cabeza del usuario. Esto es conocido como “head pose”. Dicha información permite a la aplicación saber cómo reaccionar dependiendo de la orientación y posición en cada momento.

Como aparato de salida de información, los dispositivos crean una profunda sensación de inmersión y presencia para que la sensación del usuario de estar en un ambiente sea lo más cercano posible a la realidad. Los dispositivos llevan a cabo dicha tarea de tres formas:

- Aportando un campo de visión más amplio.
- Aportando una imagen diferente para cada ojo.
- Ocultando el ambiente real entorno al usuario, para poder sumergirse completamente en el virtual.

El campo de visión que aportan estos dispositivos es mucho más amplio que el de cualquier monitor. Un monitor de unas treinta pulgadas ocupa solo 50 grados de amplitud de la vista, mientras que los dispositivos de realidad virtual ocupan entre 90 y 100 grados de amplitud de la vista. Los dispositivos consiguen esta amplitud mediante el uso de unas lentes especiales.

El dispositivo aporta dos imágenes diferentes, ya que cada ojo cubre diferentes rangos de visión. El ojo izquierdo captará más campo de visión hacia la izquierda, y el derecho captará más hacia la derecha, habiendo en medio una gran parte de la imagen que comparten ambos ojos. Una vez el cerebro recibe ambas imágenes, las fusiona y crea una única vista panorámica.

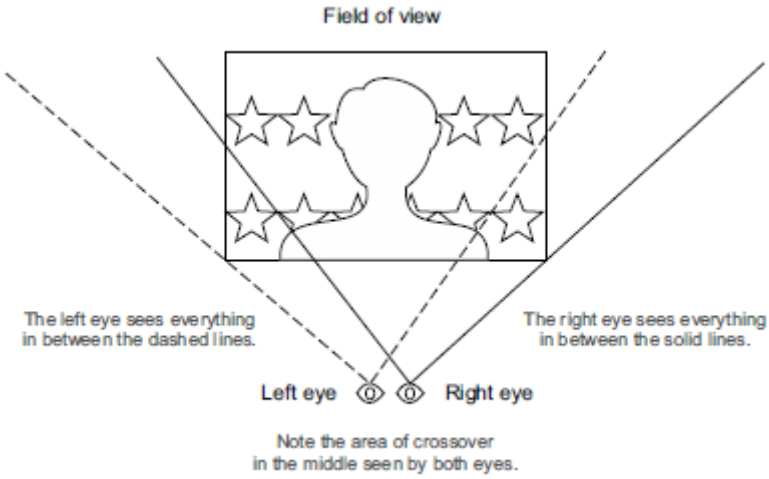

Figura 1. Campo de visión de la vista humana.

Por ello, el dispositivo crea la imagen de la izquierda con más información hacia la izquierda, y la imagen de la derecha con más información hacia la derecha, ampliando notablemente el campo de visión.

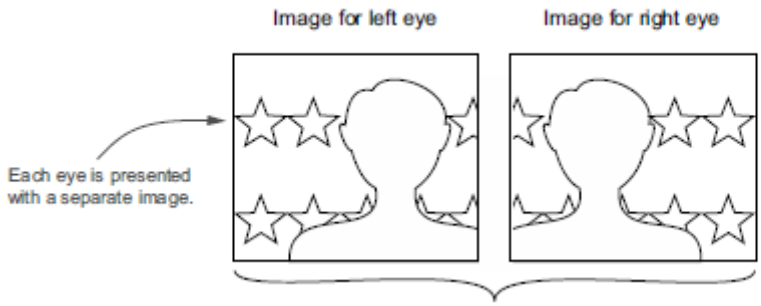

Figura 2. Campo de visión mediante el dispositivo de realidad virtual.

Otro efecto importante que crean estos dispositivos es el de distorsionar la imagen. Tienen lentes que simulan el efecto de "ojo de pez", por lo que es importante introducir una distorsión contraria para cancelar dicho efecto, y que la imagen resultante pueda apreciarse claramente.

2.3. LA INGENIERÍA CIVIL Y LA REALIDAD VIRTUAL

Los modelos geométricos tridimensionales usados para representar los trabajos de ingeniería civil y arquitectura normalmente solo muestran su forma final, sin demostrar la evolución física sufrida durante la construcción. La reproducción 3D de la construcción solamente permite observar el interior y el exterior de dicha construcción, sin poder apreciar los cambios sufridos en la geometría durante el proceso de construcción, sabiendo que dichos procesos son procesos dinámicos y que pueden sufrir diferentes cambios durante su evolución.

Las herramientas tradicionales para planificar el proceso de construcción no representan adecuadamente los aspectos espaciales y temporales. La integración de la representación geométrica junto con la planificación es la base de los modelos 4D (3D + tiempo) en la construcción. Esta cuarta dimensión combina los modelos 3D, como por ejemplo el CAD, con la progresión de la construcción con el paso del tiempo.

Mediante esta cuarta dimensión se ha demostrado que los inversores pueden entender mejor el proceso de construcción que solamente usando las

herramientas convencionales. Por todo esto, cada vez se están diseñando y desarrollando más aplicaciones basadas en la realidad virtual y en la tecnología 4D.

En relación con el seguimiento continuo de la construcción, existe la cada vez más importante metodología de trabajo BIM (Building Information Modeling). Esta metodología gestiona todos los datos de un proyecto de edificación o infraestructura desde el mismo momento en el que empieza el proceso de diseño, optimizando la gestión documental y de proyecto.

Lo importante de la metodología BIM es que trabaja con modelos tridimensionales, a los que son aplicables tecnologías como la realidad virtual. Por lo que es muy probable, que a medida que crezca la metodología de trabajo BIM, crezcan también este tipo de tecnologías, y que, en un futuro próximo, sea totalmente necesario entender y usar dichas tecnologías.

Las razones para el uso de la realidad virtual junto con la metodología BIM, es que aportan ventajas como decidir sobre la marcha las modificaciones requeridas, introducirse en la construcción virtualmente para perfeccionar el proyecto o detectar fallos, o incluso, poder realizar una mejor representación de la obra al cliente.

3. UNITY 3D

3.1. INTRODUCCIÓN

Unity es un motor de juego con una calidad profesional creado para desarrollar juegos tanto en 2D como en 3D para diferentes plataformas. Un motor de juego aporta una gran variedad de características útiles para la creación de videojuegos, por lo que un programa creado mediante dicho motor tendrá la posibilidad de elegir entre todas las características deseadas.

Como ventajas de usar Unity en lugar de otros motores de juego podemos destacar las siguientes. Permite la programación utilizando diferentes lenguajes, siendo el más destacable C#. Dispone de una modalidad de licencia gratuita que nos permite desarrollar programas desde el primer momento sin coste alguno. Soporta, además, múltiples plataformas, como por ejemplo OpenGL ES 3.0 de Android.

Además de las ventajas, Unity cuenta con las siguientes características:

- El editor de Unity permite agrupar rápidamente todas las escenas en un espacio de trabajo, mediante el uso de un editor intuitivo y fiable.

- Todos los programas creados pueden publicarse en numerosas plataformas.

- Se pueden importar modelos y animaciones realizadas con otras aplicaciones, como por ejemplo de Blender.

- Iluminación de sombras en tiempo real.

- Herramientas dedicadas a la creación de contenido 2D y 3D.

3.2. USO Y MANEJO DE UNITY 3D

En este apartado se explicarán las nociones básicas y esenciales para un mínimo entendimiento y uso de Unity.

3.2.1. PROYECTOS

Una vez ejecutado Unity, antes de entrar en el programa, aparece una ventana con dos pestañas.

La primera es la correspondiente a la gestión de proyectos. Aquí aparecerán los proyectos que se encuentren tanto en nuestro disco como en la nube, dando la posibilidad de acceder al deseado. En esta pestaña también se encuentra la opción de empezar un nuevo proyecto.

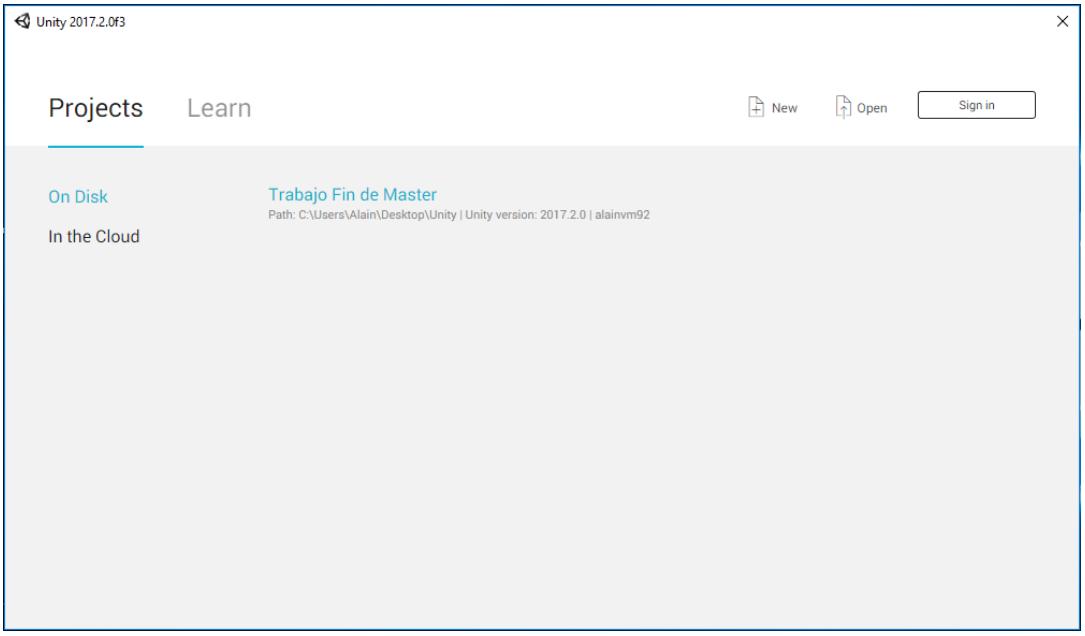

Figura 3. Pestaña proyectos de la ventana inicial.

Si se desea empezar un nuevo proyecto, aparecerá una nueva ventana en la que se tendrá que elegir el nombre para dicho nuevo proyecto, el lugar en el disco duro donde se quiera guardarlo, y si se quiere realizar un proyecto en 2D o en 3D.

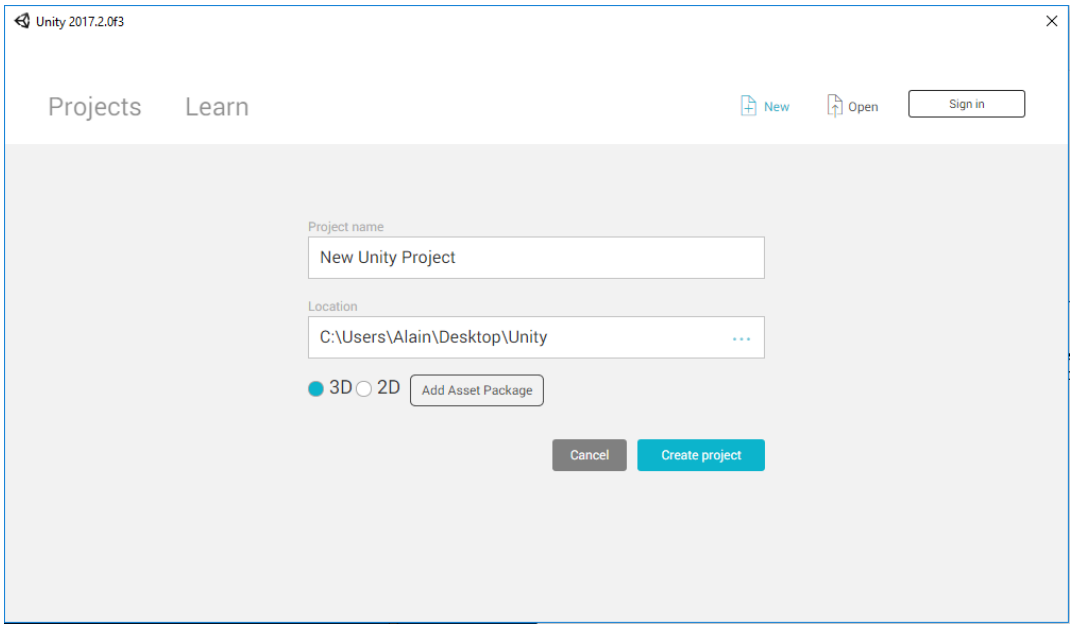

Figura 4. Creación de nuevo proyecto.

En la otra pestaña se encontrarán los diferentes tutoriales o proyectos ya creados por Unity como ayuda para entender el programa. En estos tutoriales se enseña paso a paso las nociones básicas de Unity.

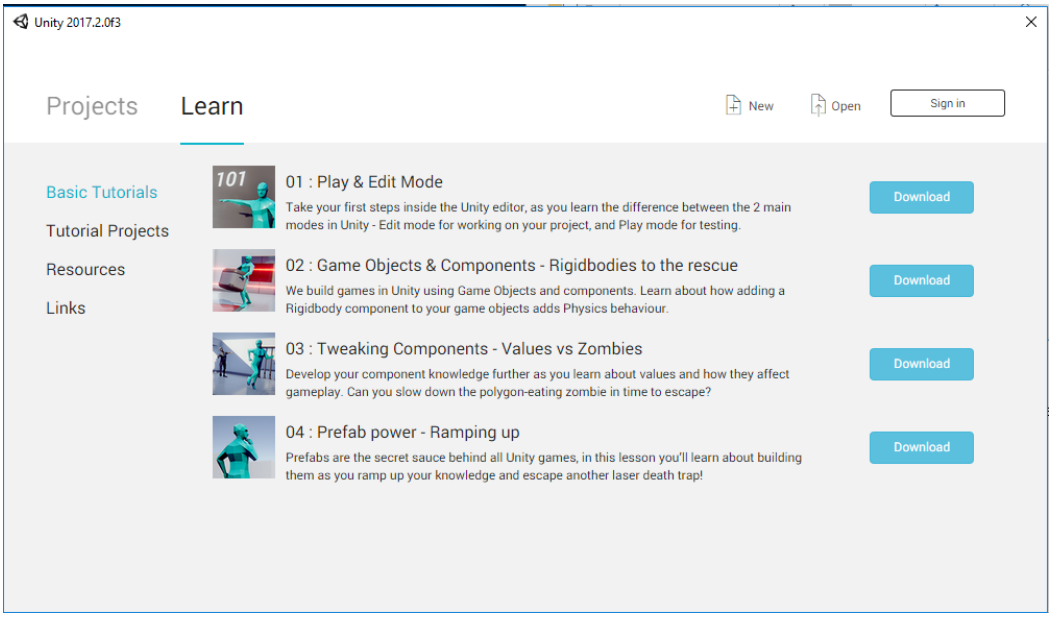

Figura 5. Pestaña "Learn" de la ventana inicial.

3.2.2. DISPOSICION DE LOS COMPONENTES

Antes de empezar a usar los diferentes componentes y elementos de Unity es importante elegir la disposición que más cómoda resulte de los mismos para después obtener un uso más fluido del programa.

A la hora de elegir dicha disposición hay que tener en cuenta los dos puntos siguientes:

- Elegir una disposición que resulte cómoda y agradable para acceder rápida e intuitivamente al elemento deseado y no perder tiempo buscándolo.

- Elegir una disposición que mejor se adapte al tipo de proyecto que se esté realizando. Si se está modelando un objeto en tres dimensiones es mejor una disposición con muchas cámaras para ver dicho objeto desde diferentes posiciones; pero si se está realizando un proyecto con muchos recursos, quizás sea mejor tener menos cámaras, para tener un mayor espacio de visión para los recursos utilizados.

Para cambiar el tipo de disposición solamente hay que hacer clic en el desplegable de las disposiciones y seleccionar la deseada.

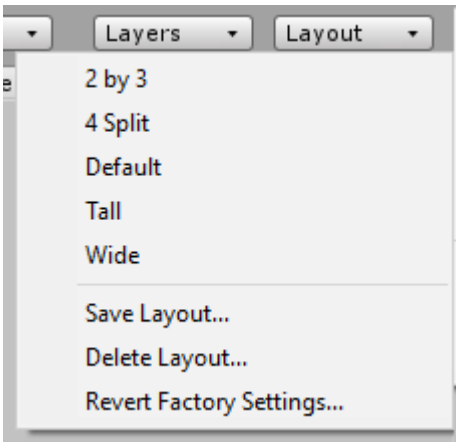

Figura 6. Desplegable de la disposición de los componentes.

3.2.3. VENTANAS

Unity cuenta con diferentes ventanas para ver los diferentes componentes del proyecto. Las cinco ventanas principales son las siguientes:

3.2.3.1. VENTANA DE LA ESCENA

En esta ventana aparecen todos los objetos visuales que coloquemos. Cada vez que se quiere trabajar con uno de dichos objetos, como por ejemplo rotarle o desplazarle, se realiza directamente en esta ventana.

Es muy importante recordar que esta ventana solo reproduce la escena actual, ya que un proyecto puede estar compuesto de diferentes escenas.

Figura 7. Ventana de la escena.

3.2.3.2. VENTANA DE JUEGO

Esta ventana representará exactamente como se reproducirá la escena actual una vez se ejecute el programa. Hay que tener en cuenta que esta ventana se verá desde el punto de vista seleccionado para reproducir dicha escena. Es decir, si la escena actual se reproduce desde una cámara que sigue a un personaje, en esta ventana se verá desde dicha cámara.

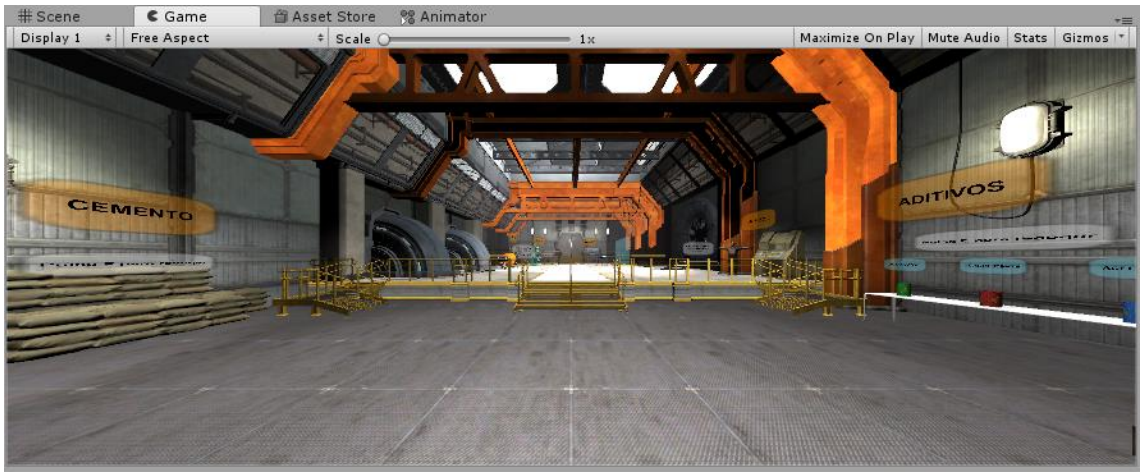

Figura 8. Ventana de juego.

Una característica muy importante de esta ventana, es que se pueden realizar los cambios deseados para ver el efecto que tienen en el programa, pero sin tener efecto en la escena, por lo que valdrá para realizar leves cambios sin importar arruinar la escena actual.

3.2.3.3. VENTANA DE JERARQUÍA

En esta ventana se pueden apreciar todos los objetos usados para crear nuestra escena. Es una ventana de gran importancia, ya que permite acceder rápidamente al objeto deseado, tanto para saber dónde está ubicado, como para saber cuáles son sus características.

También permite saber si alguno de los objetos esta creado por uno o más complementos y poder acceder a dichos complementos.

Figura 9. Ventana de jerarquía.

3.2.3.4. VENTANA DE PROYECTO

En la ventana de proyecto aparecen todos los recursos utilizados, no solamente los de la escena actual; y además de los elementos físicos, también aparecen scripts, audios, animaciones… En esta ventana se pueden encontrar las diferentes escenas del proyecto, por lo que si se desea cambiar de escena se puede hacer desde esta ventana.

Esta ventana contiene una segunda columna, en la que aparecerán todos los complementos que contenga la carpeta seleccionada.

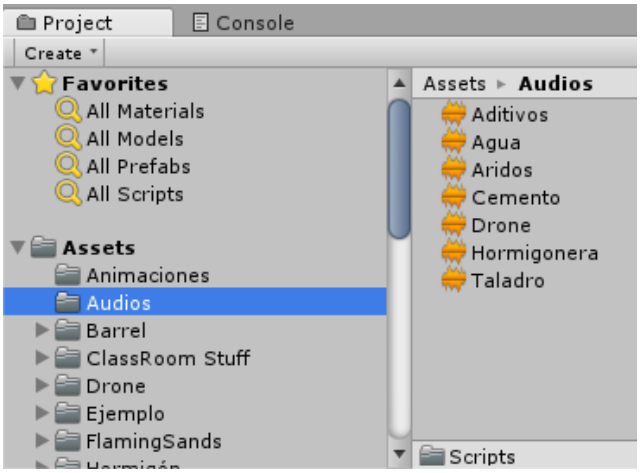

Figura 10. Ventana de proyecto.

3.2.3.5. VENTANA DEL INSPECTOR

En el inspector aparecen todas las características del objeto seleccionado. Estas características van desde el tamaño o la posición del objeto, hasta los diferentes scripts que contiene dicho objeto.

Una de las ventajas de esta ventana es que se pueden realizar los cambios deseados desde la ventana en cuestión. Es decir, se puede cambiar la posición de un objeto cambiando el valor de la posición en la misma ventana.

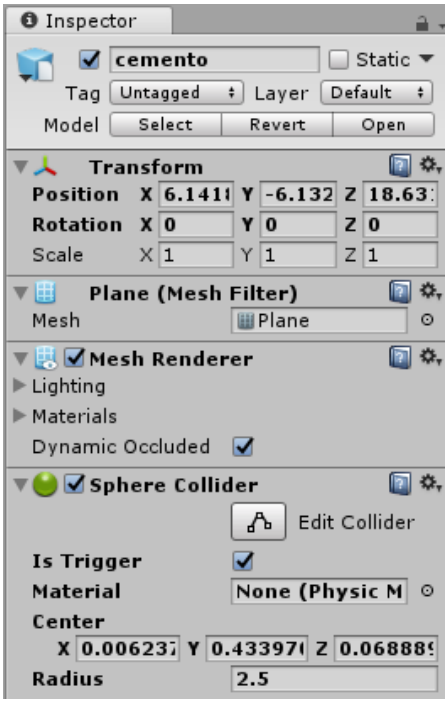

Figura 11. Ventana del inspector.

3.2.4. HERRAMIENTAS DE TRANSFORMACIÓN

Las herramientas de transformación son de gran importancia, ya que permiten modificar ciertos parámetros físicos de objetos directamente en la ventana de la escena, sin tener que ir a la ventana del inspector. Hay cinco herramientas de transformación diferentes:

Figura 12. Herramientas de transformación.

- La primera es la mano, que permite moverse por la escena; tanto hacer zum como desplazarnos lateral o verticalmente. Si se mantiene pulsado el “Alt” también permite orbitar alrededor de la escena.

- La segunda es la herramienta de translación. Dicha herramienta permite mover objetos en la escena.

- La tercera es la herramienta de rotación. Es igual que la anterior, pero en lugar de permitir mover el objeto, permite rotarle en cualquier dirección.

- La cuarta es la herramienta de la escala. Permite escalar el objeto en cualquiera de las tres direcciones principales.

- La quinta y última herramienta es una fusión de algunas de las herramientas anteriores, ya que permite mover, escalar y rotar el objeto seleccionado en la escena.

3.2.5. BOTONES DE REPRODUCCIÓN

Encima de la ventana de escena existe un panel que contiene tres botones.

El primero de todos es el botón de Play, que haciendo clic en el, reproduce la escena actual. Es muy importante tener en cuenta que solamente reproduce la escena actual, por lo que si el proyecto en cuestión, tiene más de una escena con un determinado orden, tendremos que hacer clic en el botón de Play estando en dicha primera escena.

El segundo botón es el de pausa, que nos permite parar la escena en el momento deseado. Este botón es muy útil sobre todo en escenas con muchas animaciones, en las que se quiera pararlas para realizar pequeños cambios.

El tercer y último botón es el de avanzar escena, que servirá para avanzar en la escena actual.

Figura 13. Botones de reproducción.

3.2.6. IMPORTAR PAQUETES

Una de las herramientas más útiles de Unity es la capacidad de importar diferentes elementos ya creados. Se pueden importar cualquier tipo de elementos; desde entornos ya construidos, hasta personajes prefabricados con sus propias animaciones de movimiento.

Estos paquetes se pueden encontrar tanto en la Asset Store de Unity como por internet; y aunque algunos sean de pago, existen otros muchos que son gratuitos. El poder importar estos paquetes ya creados es un gran ahorro de tiempo a la hora de realizar el proyecto, ya que casi siempre se podrá encontrar alguno objeto muy similar al necesitado, y de esta manera ahorrarse el tener que crearlo.

Otra de las ventajas es la sencillez para importar estos paquetes. Si se cogen de la Asset Store, a la hora de descargarlo, el mismo programa nos da la opción de importarlo directamente. En el caso de haberlo descargado de otro sitio y tener que importarlo manualmente, es necesario ir a la barra del menú, seleccionar la pestaña Asset y después hacer clic en “Import New Asset…”.

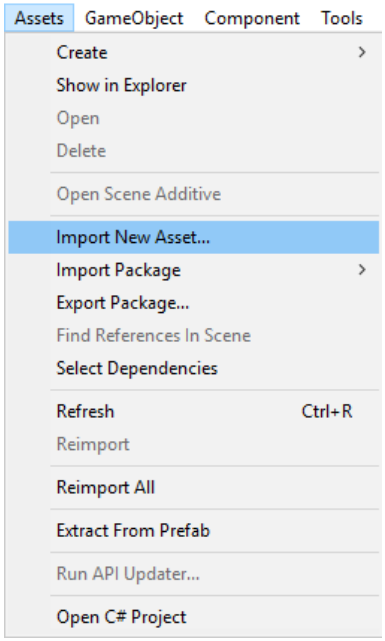

Figura 14. Pestaña para importar paquetes.

3.2.7. PROGRAMACIÓN

Una de las ventajas de Unity es que usa un lenguaje de programación conocido como es el C#. Además, el propio Unity cuenta con una gran cantidad de funciones para hacer mucho más fácil e intuitiva la programación.

Hay que tener en cuenta, que, dependiendo del tipo de proyecto, la programación puede ser una gran parte de dicho proyecto. Toda interacción entre diferentes objetos requiere una cierta programación, por lo que, si en el proyecto a realizar hay mucha interacción, también habrá una gran cantidad de programación.

La programación en Unity funciona por scripts. Por ello, para programar un objeto, primero se escribe el código requerido en un script y, después, se adjunta al objeto deseado. Esto también da la ventaja de poder aplicar un mismo script a una cantidad indefinida de objetos.

En caso de existir algún error en el código, el propio Unity no dejará inicializar la escena, diciendo cuál es el error en cuestión y dónde se produce.

3.3. EL USO DE LA REALIDAD VIRTUAL EN UNITY

Unity es un programa que ofrece una gran facilidad para el uso de la realidad virtual. Al igual que ocurre con los otros tipos de elementos, también se pueden descargar paquetes prefabricados de realidad virtual ya pre-configurados. Es decir, no es necesario tener que programar ni configurar nada para empezar a usar la realidad virtual en Unity.

El único problema que puede aparecer, es que dependiendo de la versión de Unity, quizás sea necesario activar la opción de realidad virtual en el programa. Para activar dicha opción, en el caso de ser necesario, en la barra de herramientas en el menú de *Edit*, se entra en *Project Settings*, después en *Player*, y una vez aparezca la configuración en la ventana del inspector se hace clic en la pestaña *XR Settings*.

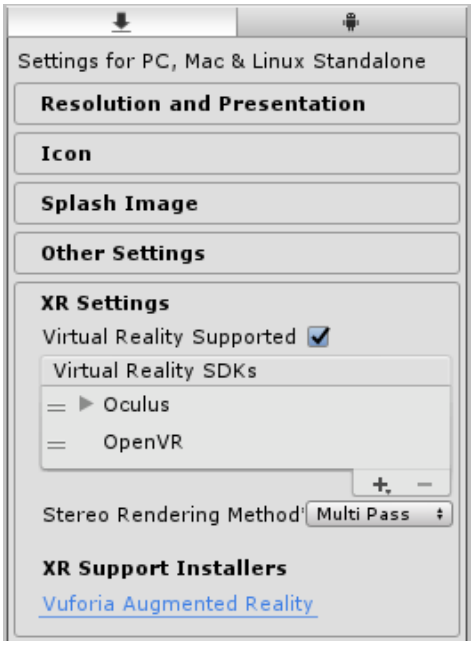

Figura 15. Ventana para la configuración de la realidad virtual.

Al activar esta opción, Unity ofrece todavía más facilidades en relación con la realidad virtual, ya que automatiza los siguientes procesos:

- Se adapta el campo de visión y se activa automáticamente el seguimiento de cabeza.

- Las optimizaciones suceden automáticamente para facilitar el dibujar *frames* dos veces, uno para cada ojo.

- Ajusta las matrices de vista y proyección para tener en cuenta el campo de visión, y de esta manera, no requerir el uso de dos cámaras para las visualizaciones estereoscópicas.

4. EJEMPLO DE PROGRAMA PARA EL USO DE LA REALIDAD VIRTUAL EN UNITY

4.1. INTRODUCCION

En el presente trabajo se ha elegido hacer un programa didáctico en lugar de virtualizar una obra. La razón principal es demostrar la gran variedad de posibilidades que ofrecen este tipo de programas y en general la realidad virtual.

Lo que en todo momento se ha buscado es amenizar el proceso didáctico mediante el uso de la realidad virtual, sabiendo que el hecho de interactuar con el entorno y los diferentes elementos puede captar la atención del usuario en un mayor grado.

Independientemente de lo anterior, la finalidad del programa es didáctica. En este caso se busca aprender a hacer hormigón, ofreciendo una explicación de los materiales necesarios, y de las diferentes características que tendrá dicho hormigón dependiendo de la cantidad de los materiales utilizados.

El desarrollo del programa se ha hecho pensando en un posible futuro uso del mismo en la escuela. Así se ha diseñado para que su uso sea lo más sencillo e intuitivo posible.

La explicación de la realización del programa se hará siguiendo el mismo orden que se utilizó para la creación del mismo.

4.2. EL ENTORNO

4.2.1. ESTRUCTURA GENERAL Y ESCENA

Como ya se ha comentado con anterioridad, los proyectos de Unity se basan en diferentes escenas. Por lo que lo primero es elegir y descargar la escena y su respectivo entorno, para que no sea necesario crear una desde cero.

En el caso de este proyecto se ha elegido el paquete llamado *Robot Lab*, que se encuentra en la *Asset Store*. Una vez localizado se descarga, y al ser de la *Asset Store*, en cuanto finaliza la descarga el propio programa te da la opción de importarlo a Unity.

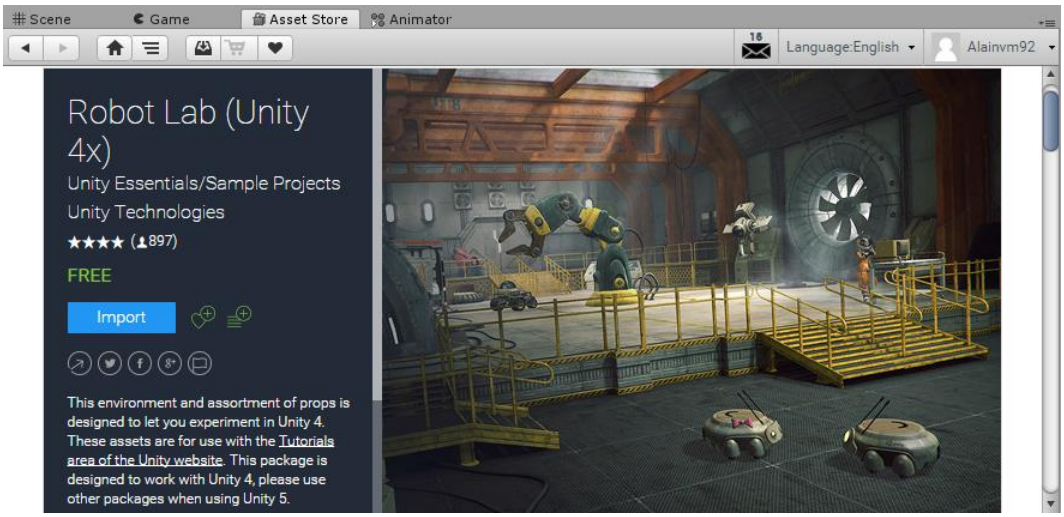

Figura 16. Paquete "Robot Lab".

Figura 17. Ventana para importar el paquete "Robot Lab".

Después de descargarlo e importarlo aparecerán diferentes carpetas en la ventana *Project*. Como lo que se quiere es una escena, se busca la carpeta *Scene*, y dentro de la misma aparecerán dos escenas diferentes, *Empty* y *RobotLab*. Solo se desea el entorno, y, además, durante este proyecto se van a añadir cosas nuevas, por lo que se selecciona la escena *Empty* (vacía).

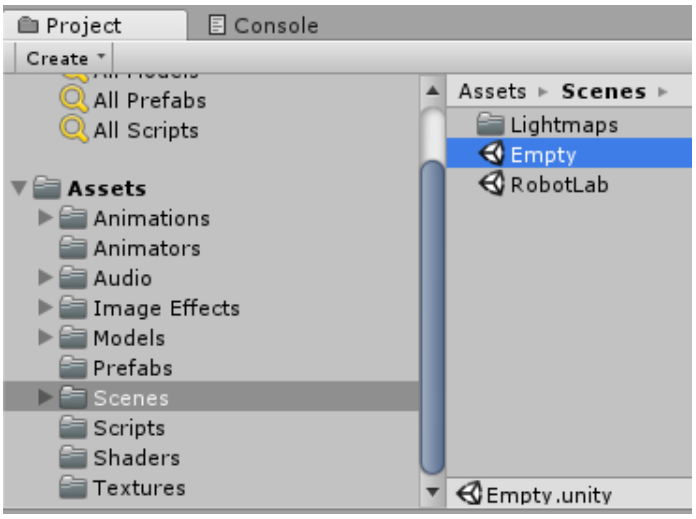

Figura 18. Escena del paquete "Robot Lab" seleccionada.

Una vez se haya hecho doble clic en *Empty*, en la ventana *Scene* se podrá observar que han aparecido diferentes objetos, eso significa que ya se está trabajando en dicha escena. A partir de este momento, todo el trabajo realizado será en esta escena hasta que se cree una nueva.

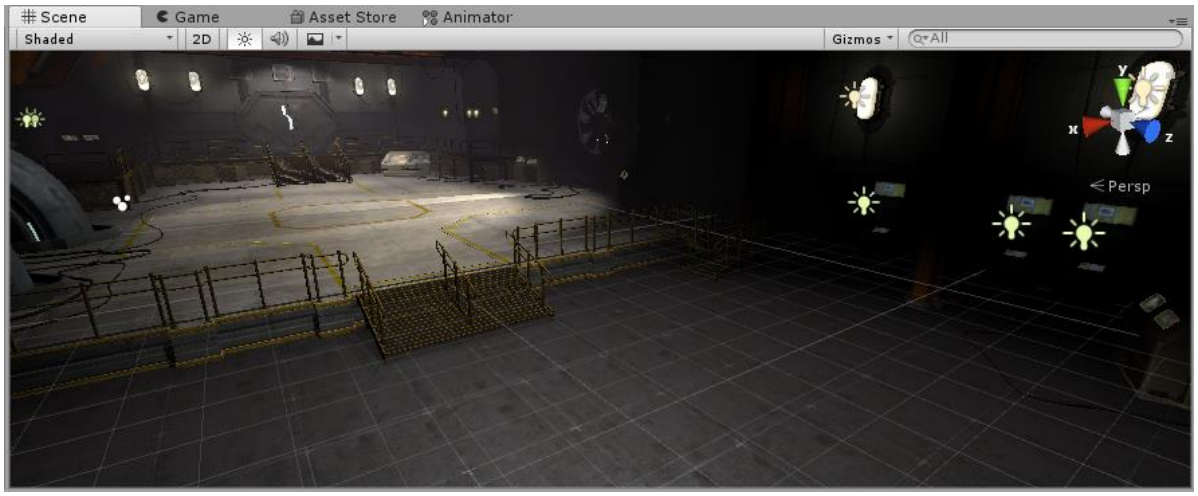

Figura 19. Vista de la escena seleccionada.

4.2.2. OBJETOS IMPORTADOS

El proyecto se basa en la interacción con diferentes objetos, por lo que una vez se ha obtenido el entorno, se puede proceder a colocar los objetos deseados.

Para la colocación de los mismos, se ha seguido el mismo proceso para todos, excepto para la hormigonera, que se explicará más en detalle. En la explicación del proceso, se usarán los sacos de cemento como ejemplo.

Lo primero es pensar que objeto se quiere colocar, porque los objetos no se crearán, si no que se importarán para no tener que crearlos; y, además, se ha optado siempre por los paquetes gratuitos, por lo que se reduce en gran medida el número de opciones.

En el caso del cemento, se ha optado por buscar unos sacos que representen sacos de cemento. Al igual que con la escena, se busca en la *Asset Store* un paquete que sean sacos. El proceso de descarga e importación es el mismo que con el entorno.

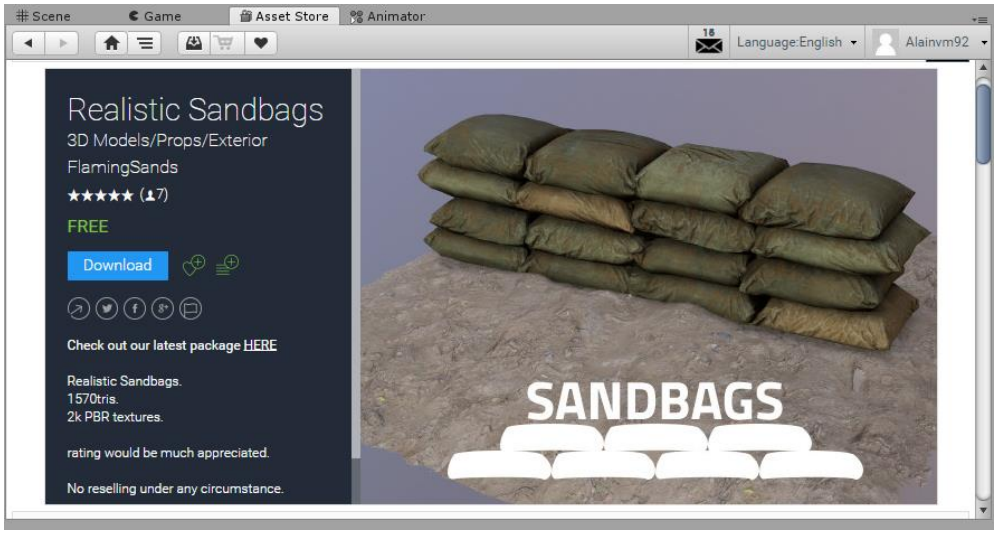

Figura 20. Vista en la Asset Store del paquete seleccionado.

Una vez descargado e importado, al igual que con el paquete anterior, aparecerán nuevas carpetas en la ventana de proyecto. Pero esta vez, no se busca una escena, sino, solamente el objeto en sí. Por lo que hay que encontrar un archivo que sea un objeto. Para aplicarlo a la escena con arrastrarlo desde la carpeta hasta la ventana de *Scene* es suficiente.

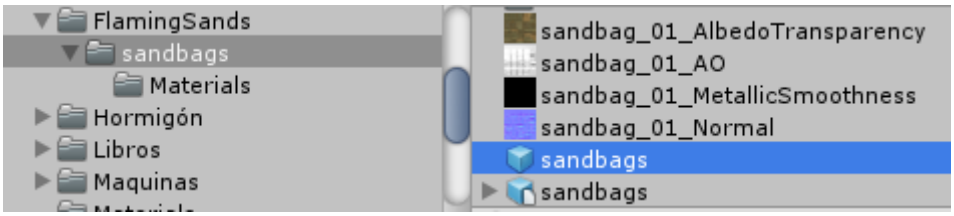

Figura 21. Diferentes elementos del paquete importado.

El siguiente paso es ajustarlo a las características deseadas. Es muy probable que el objeto no tenga la orientación o tamaño deseado, por lo que hay que modificarlo, bien usando las herramientas de modificación, o mediante la ventana del inspector.

Los pasos para colocar los demás objetos son idénticos. La mayor diferencia está en el último paso, ya que cada objeto necesita un tamaño y una posición determinada. Incluso habrá objetos que se colocarán encima de otros objetos, por lo que primero será necesario colocar algunos de ellos, para colocar los siguientes.

4.2.2.1. HORMIGONERA

El caso de la hormigonera es diferente al resto. La razón es que no se encontró ningún paquete de una hormigonera que fuese gratuito. La solución ha sido buscar un camión hormigonera, que, si es gratuito, y quedarse solo con la parte de la hormigonera.

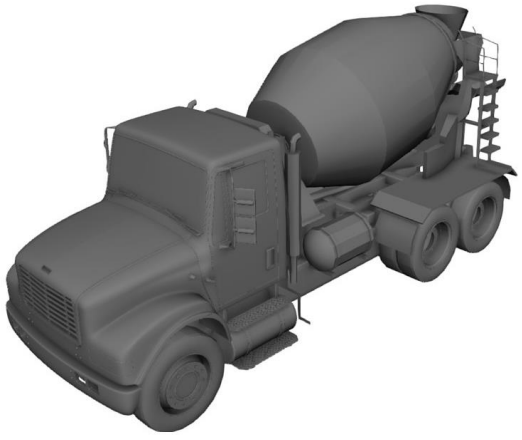

Figura 22. Objeto 3D del paquete descargado.

Para realizar dicha operación ha sido necesario usar el programa informático Blender. Este programa permite abrir, modificar los objetos 3D y después, exportarlos a diferentes programas.

En el caso del camión, primero se ha separado todos lo subelementos que lo componen, para poder borrar los no deseados. Una vez que se tiene solo la parte deseada, en este caso la de la hormigonera, se exporta el archivo guardado a Unity.

Para exportarlo, simplemente hay que ir a la barra de herramientas y seleccionar File -> Export -> FBX. Es importante exportarlo como archivo.fbx para que después Unity pueda reconocerlo.

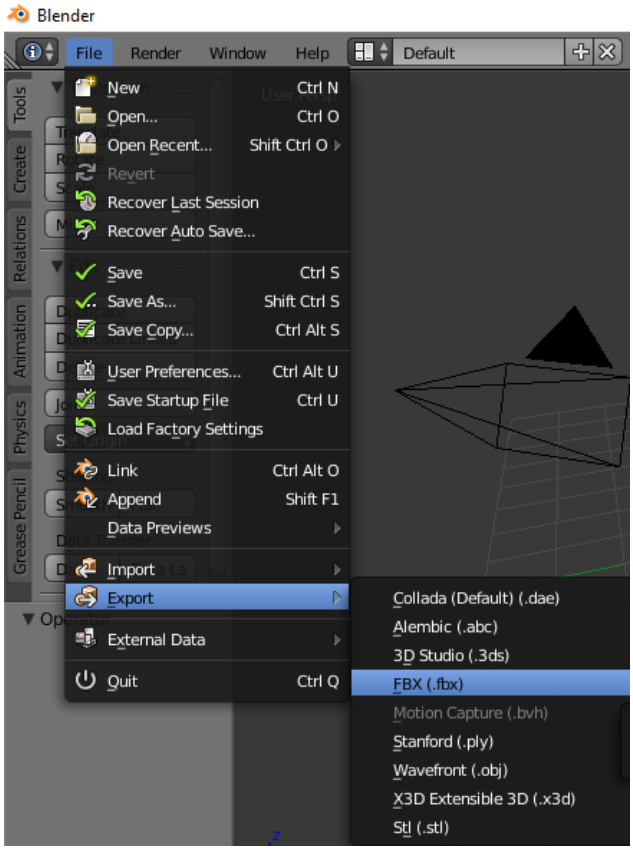

Figura 23. Pestaña "File" del programa Blender.

Una vez Unity abre el archivo, el procedimiento para colocarlo es igual que con el resto de los objetos.

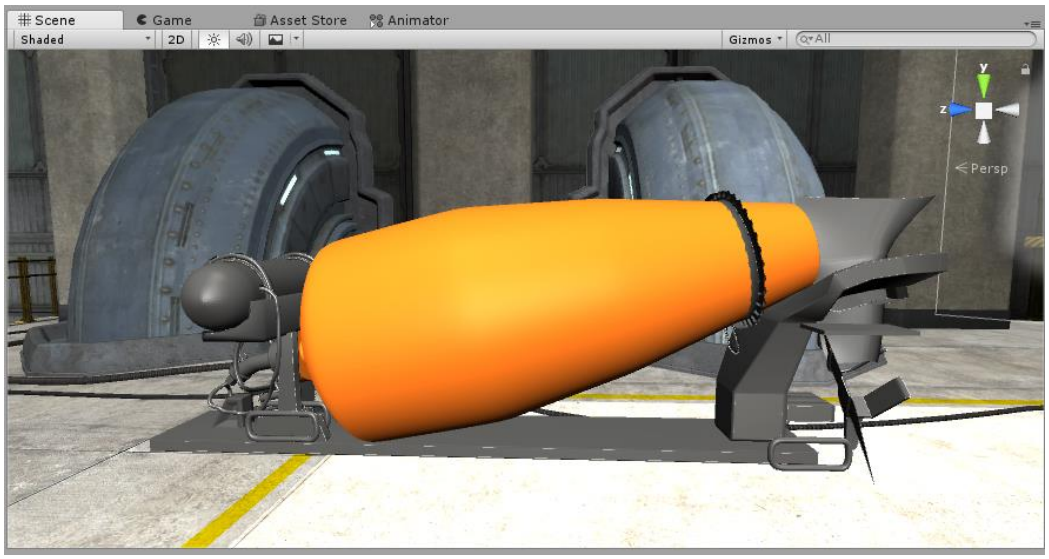

Figura 24. Hormigonera resultante en el Unity.

4.2.3. OBJETOS CREADOS

Como se ha comentado con anterioridad, la mayoría de los objetos han sido importados, pero existen unos pocos que han tenido que ser creados, ya que no pueden ser importados. Estos objetos son los siguientes:

4.2.3.1. LUCES

Al ser una escena prefabricada, la distribución de luces que contiene puede no ser la deseada y puede ocurrir que los lugares donde colocamos los nuevos objetos no tengan la iluminación deseada; o simplemente, que a objetos importantes se les quiera aplicar una mayor iluminación para que atraigan la atención del usuario.

Al igual que cualquier otro objeto, las luces se crean haciendo clic en la barra de herramientas en el menú *GameObject*. Una vez se ha desplegado el menú se hace clic en *Light*, y después, en *SpotLight*.

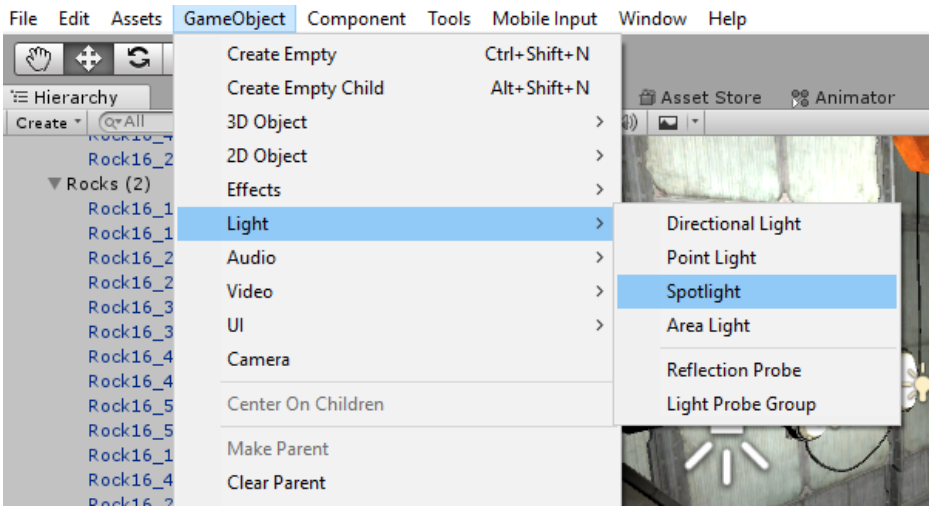

Figura 25. Menú para la creación de las luces.

Se ha elegido el tipo *Spotlight* porque son luces direccionales, pero no llegan a ser tan intensas como las *Directional Light*. Como son direccionales, lo primero es ajustarle los valores de rotación y de posición, para alumbrar al objeto deseado. Además de dichos valores, hay otros tres de gran importancia:

- *Range*: indica la distancia máxima efectiva de la iluminación de dicha luz. A mayor valor, mayor será la distancia a la que llegue la luz.

- *Spot Angle*: este valor indica la apertura de la luz. A mayor valor, mayor será el ángulo que abarca dicha luz.

- *Intensity*: mediante este valor se ajusta la intensidad de la luz, es decir, a mayor valor mayor será la intensidad de la luz.

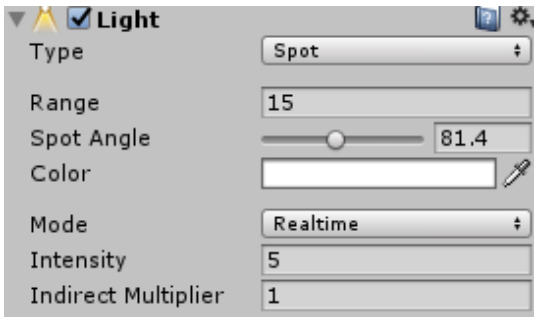

Figura 26. Características de la luz.

Estos valores son de gran importancia, ya que dependiendo del valor que les asignemos conseguiremos darle más o menos importancia al punto iluminado. Incluso también sirve para poder alumbrar más de un objeto a la vez, y no llenar la escena de luces.

4.2.3.2. CANVAS

Es uno de los *GameObjects* con mayor importancia de Unity, ya que todos los elementos UI que se creen deben de ser hijos de algún *Canvas*. Por lo que cada vez que se ha querido crear alguna consola con botones, o algún panel informativo con texto, primero ha sido necesaria la creación de uno.

Al tratarse de un objeto, para su creación hay que ir al menú *GameObject*, después a UI, y dentro de la opción de menú UI se selecciona el objeto *Canvas*.

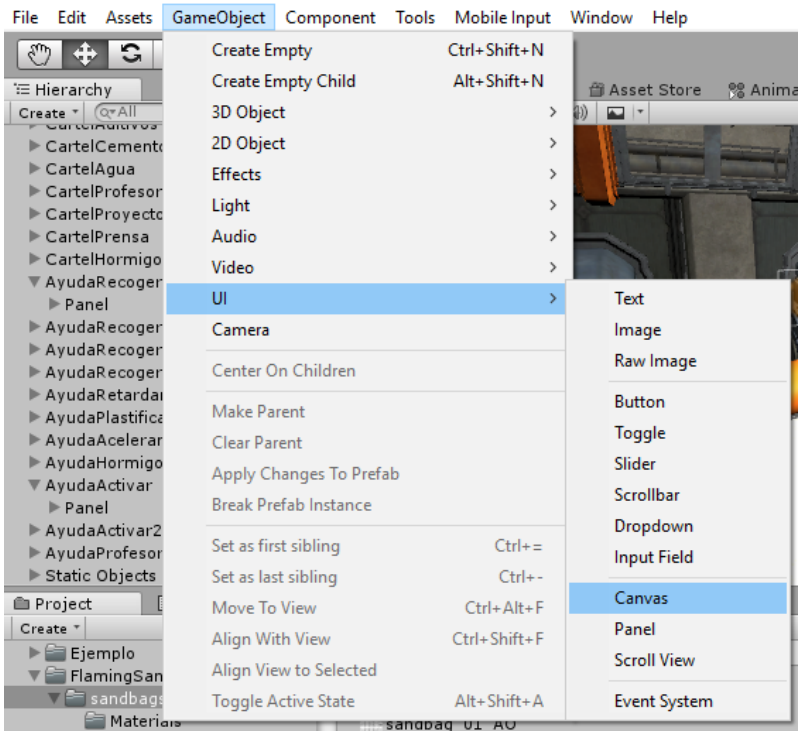

Figura 27. Pestaña para la creación del Canvas.

En este trabajo se han usado los siguientes elementos de interfaz de usuario:

4.2.3.2.1. BOTONES

Como es un elemento de interfaz para su creación hay que ir al menú *GameObject*, después a UI, y dentro de la opción de menú UI se selecciona el objeto *Button*.

La función de estos botones es la de realizar una acción cada vez que se hace clic en ellos, como, por ejemplo, puede ser activar un audio. Hay que tener en cuenta que es necesaria la programación para el funcionamiento de los botones.

Figura 28. Resultado de los botones en Unity.

4.2.3.2.2. SLIDER

Al igual que los botones, se trata de otro elemento de interfaz, por lo que para su creación hay que ir al menú *GameObject*, después a UI, y dentro de la opción de menú UI se selecciona el objeto *Slider*.

Estos elementos dan la posibilidad de seleccionar un valor determinado. En este caso, se ha utilizado para darle un valor determinado a la cantidad utilizada de los objetos para la creación del hormigón. Al igual que los botones, necesitan ser programados para que funcionen.

Figura 29. Resultado de los Sliders en Unity.

4.2.3.2.3. TOGGLE

Como es otro elemento de interfaz, los pasos para la creación son ir al menú *GameObject*, después a UI, y dentro de la opción de menú UI se selecciona el objeto *Toggle*.

En este caso, la función de estos elementos es la de decir “si o no”. En este trabajo, sirven para utilizar o no un determinado elemento en la creación del hormigón. Al igual que con los anteriores elementos de interfaz, necesitan ser programados para su funcionamiento.

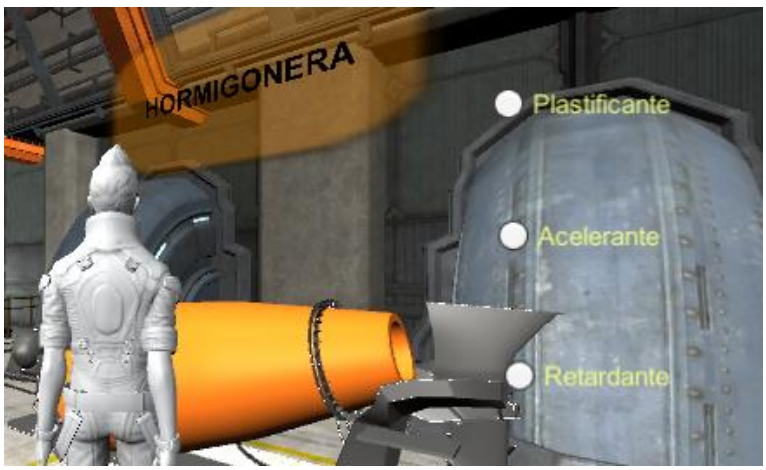

Figura 30. Resultado de los Toggles en Unity.

4.2.3.2.4. PANEL

Aunque los pasos requeridos para su creación son los mismos que para los elementos anteriores, ir al menú *GameObject*, después a UI, y dentro de la opción de menú UI seleccionar el objeto *Panel,* su función es meramente decorativa. Este elemento simplemente le da profundidad al *Canvas* para que pueda apreciarse claramente.

Como su función es decorativa, estos elementos no necesitan programación alguna para su funcionamiento.

Figura 31. Resultado de los paneles en Unity.

4.2.3.2.5. TEXT

Todo texto que quiera escribirse en cualquiera de los anteriores elementos necesita un elemento de *Text*. Es necesario, tanto en los botones, *sliders* o *toggles* para ponerles un nombre, así como en los paneles para que se pueda mostrar la información deseada.

Estos elementos se crean igual que los anteriores, pero haciendo clic en *Text*, y tampoco necesitan ningún tipo de programación, ya que su función es la de informar.

4.2.3.3. PERSONAJES

Los personajes en Unity se consideran objetos. La diferencia con el resto de los objetos, es que los personajes son objetos dinámicos. Existen dos tipos diferentes, los que están controlados por el ordenador, y los que están controlados por un humano.

Los que están controlados por el ordenador son conocidos como personajes no jugadores. Estos personajes tendrán equipadas diferentes animaciones de moverse e interactuar con el entorno. En este proyecto, tanto el profesor como el *drone* son personajes no jugadores.

El personaje que controlará el usuario para navegar e interactuar en el tutorial será un personaje jugador, es decir, lo manejará una persona y no el ordenador. Para controlar dicho personaje son necesarios los llamados controladores de personaje.

Al igual que con otros elementos, Unity cuenta con diferentes controladores de personajes prefabricados. Existen dos tipos diferentes dependiendo de la perspectiva; el controlador de jugador en primera persona, y el controlador de jugador en tercera persona.

En este proyecto, como es en realidad virtual, se ha usado el controlador en primera persona. Al igual que con el resto de elementos, para colocarlo es necesario arrastrar el elemento prefabricado a la ventana de la escena.

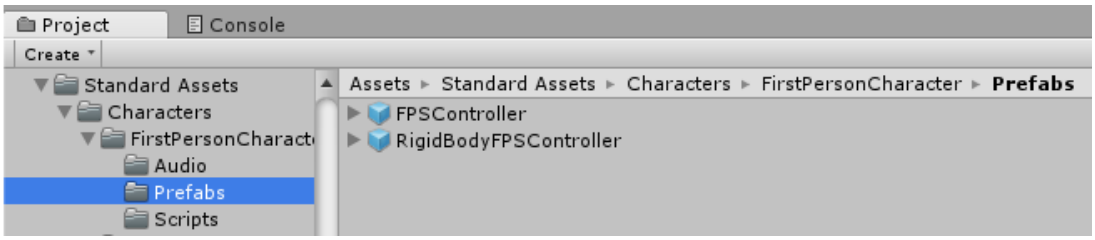

Figura 32. Elementos prefabricados del controlador.

4.2.3.3.1. OVR (Oculus Virtual Reality)

Como el dispositivo de realidad virtual que va a utilizarse es el Oculus Rift, se necesitaran los controladores correspondientes a dicho dispositivo. Es por ello que las siglas OVR (*Oculus Virtual Reality*) hacen referencia al Oculus Rift.

Aunque el OVR se trate de un elemento descargado y posteriormente importado, su explicación se realiza en este apartado debido a que funciona conjuntamente con el anteriormente explicado controlador de personaje.

El OVR, al igual que otros elementos, al ser importados contienen diferentes carpetas. En este caso interesa la carpeta *Prefrabs* que contiene los elementos prefabricados, es decir, están preparados para funcionar de inmediato.

Dentro de la carpeta *Prefabs*, el elemento a utilizar es el *OVRCameraRig*. La razón de usar este y no otro, es que en el proyecto ya se está usando un controlado de personaje, por lo que solamente con usar la cámara será suficiente. Para añadirlo, al igual que en casos anteriores, se arrastrará desde la ventana *Project* a la ventana de la escena. Después, será necesario colocarla y orientarla en el lugar y posición correcta.

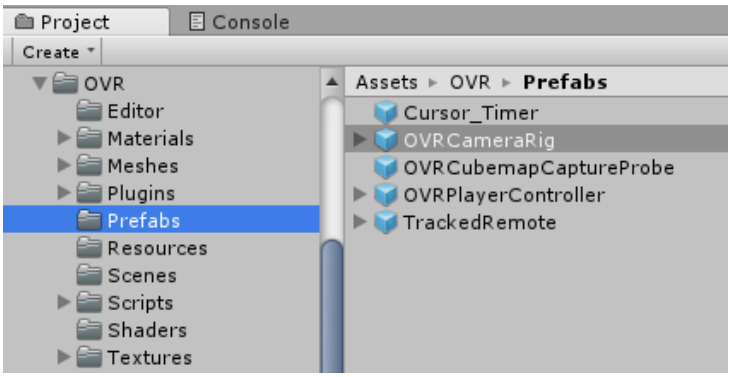

Figura 33. Elementos prefabricados del OVR.

Como ya se ha comentado, este elemento es una cámara, por lo que antes de utilizarla es necesario asegurarse de que no hay ninguna otra cámara principal activada, es decir, esta será la única activada de toda la escena. Una vez realizado dichos pasos, al darle al Play, la escena se reproducirá desde la *OVRCameraRig*.

Figura 34. Vista de la escena en realidad virtual.

4.2.4. ANIMACIONES Y AUDIOS

En este apartado se explicará solamente como se han creado los audios y las animaciones, más adelante se hablará sobre su aplicación y la función que desarrollan en el trabajo.

Existe el caso del *drone* que es algo más complejo que el resto de casos, por lo que se le dedicará un apartado para explicarlo en detalle. Para los demás casos, la creación de todos audios y animaciones se ha desarrollado de igual manera.

4.2.4.1. ANIMACIONES

Las animaciones permiten que los objetos realicen ciertos movimientos, como por ejemplo desplazarse o rotar, en el desarrollo de la aplicación. Para la creación de una animación, hay que ir al menú *Window*, y dentro de este, hacer clic en *Animation*.

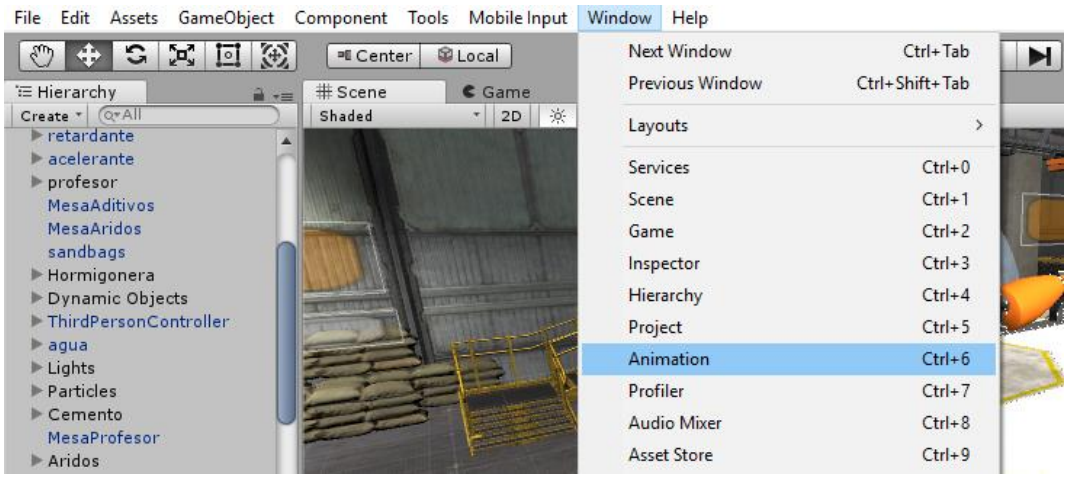

Figura 35. Menú para la creación de animaciones.

A continuación, se abrirá una nueva ventana. En el caso de que no haya ningún objeto seleccionado, el programa nos dirá que es necesario seleccionar uno, por lo que hay que seleccionar el objeto que se desea animar. Una vez seleccionado, se hace clic en el botón *Create*, y esto hará que se genere un archivo de tipo animación.

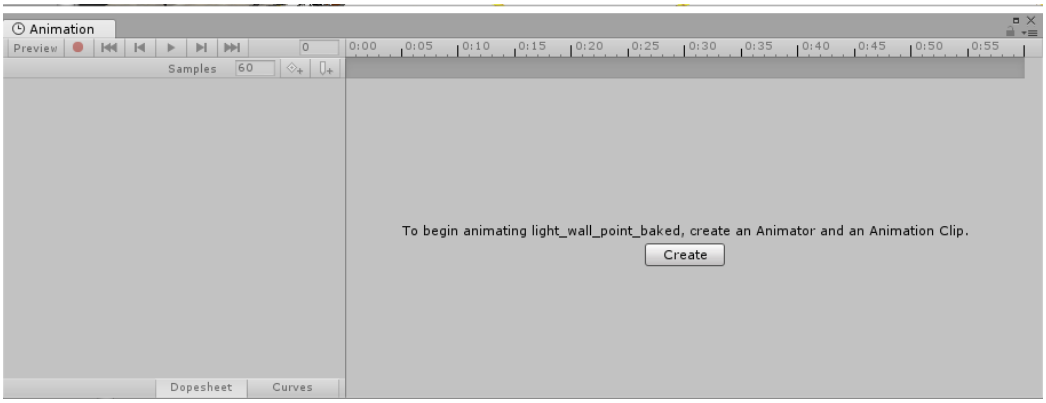

Figura 36. Ventana para la creación de animaciones.

Después de crear el archivo, en la ventana de animación aparecerán nuevas opciones. A la izquierda, habrá aparecido un botón que se llama *Add Property*. Dicho botón nos permite seleccionar las propiedades que se quieran animar. Por ejemplo, en el caso de la prensa, como solo se ha querido que rote, y no que se desplace, se selecciona la propiedad *rotation*.

La parte derecha de la ventana describe la velocidad que se desea para la animación.

Explicado mediante un ejemplo, si se quiere que un objeto se desplace, en la parte de la izquierda se dice cuanto y en que eje se quiere que se desplace, y en la parte derecha se indica, cuanto se quiere que se tarde en realizar dicho desplazamiento. El proceso final es algo más complejo, ya que una animación va a estar formada por más de un desplazamiento, por lo que el tiempo también va a estar fragmentado para realizar todas y cada uno de esos desplazamientos.

En la siguiente foto se puede apreciar cual ha sido la propiedad a modificar, y como se ha fragmentado el tiempo.

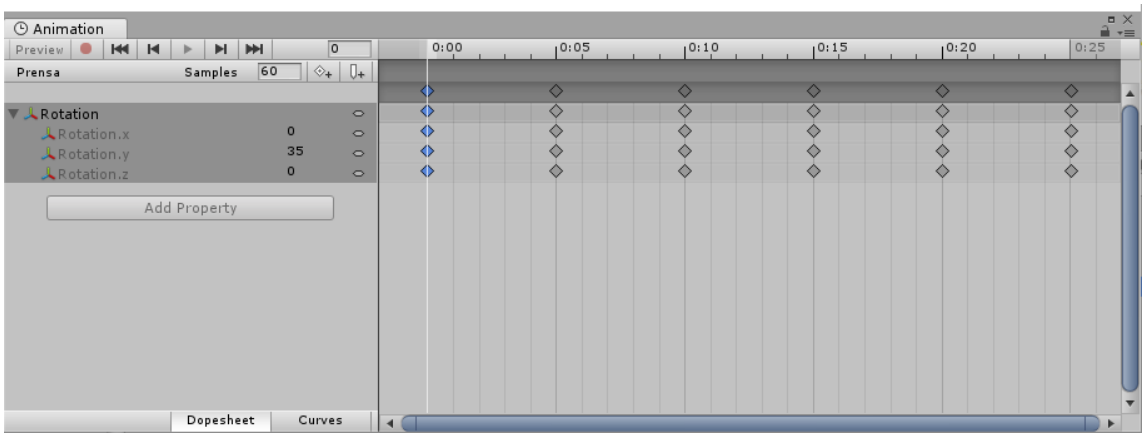

Figura 37. Imagen resultante de una animación creada.

Después de crear la animación, en el objeto a animar habrá aparecido un nuevo componente en la ventana del inspector llamado *Animation*, en el que estará añadida la animación creada.

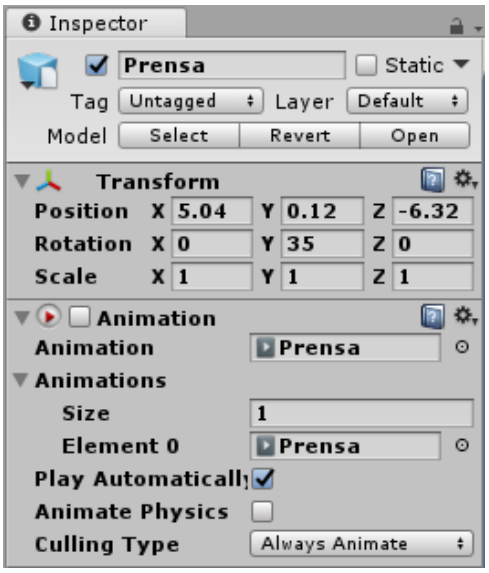

Figura 38. Vista en el inspector de la pestaña Animation.

4.2.4.2. AUDIOS

Al contrario que las animaciones, los audios no se han creado en Unity, si no que se han utilizado las siguientes dos opciones. La primera ha sido descargar de internet diferentes clips de audio ya creados, y la segunda, utilizar el programa Audacity para grabarlos.

En internet existe una gran variedad de páginas especializadas para la descarga de clips de audio, por lo que en las mismas se podrán encontrar diferentes audios. Además, algunas de ellas, ofrecen seleccionar el formato en el que se desea dicho clip de audio.

Por otro lado, el programa Audacity es un programa especializado en la creación y modificación de audios. La razón de usar este programa, es que hay diferentes audios que han sido grabados y otros que han sido modificados.

Una vez descargados o creados, Unity ofrece la posibilidad de importarlos y guardarlos en el programa. Para reproducir dichos audios, el siguiente paso es crear un *Audio Source*.

Para crear el *Audio Source* se selecciona el objeto que se quiera que reproduzca los audios. Una vez seleccionado, en la ventana del Inspector, se baja el *scroll* todo lo posible, y se hace clic en el botón *Add Component*.

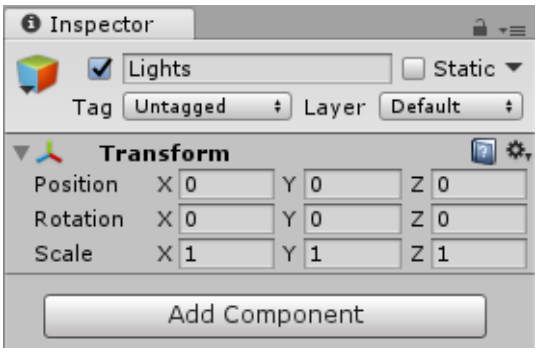

Figura 39. Botón del inspector para añadir un componente.

Al hacer clic, se abre una nueva pestaña con diferentes componentes. En este caso se hace clic en el componente *Audio*, y volverá a parecer un nuevo menú con diferentes opciones. En ese nuevo panel se hace clic en *Audio Source*.

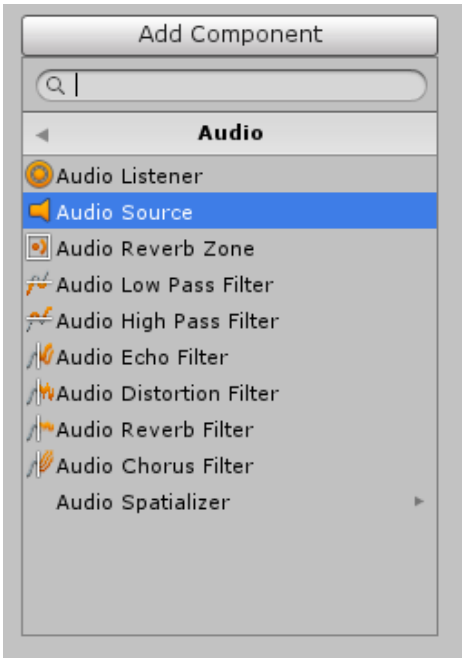

Figura 40. Menú para la creación de un Audio Source.

Una vez seleccionado, se podrá observar que, en el objeto, en la ventana del inspector, ha aparecido un nuevo componente, parecido al ya nombrado anteriormente *Animation*. Además, su uso es parecido, ya que en dicho componente hay que poner el clip de audio deseado para que se reproduzca.

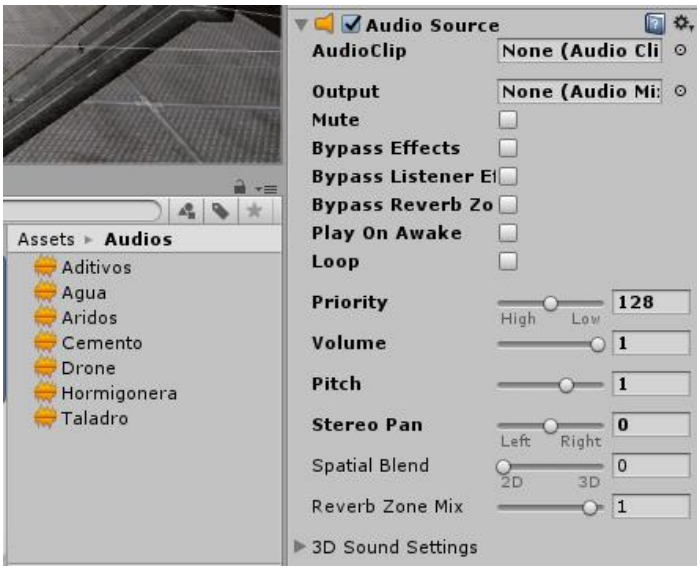

Figura 41. Componente de Audio Source.

Para colocar el clip deseado, simplemente con arrastrar dicho clip desde la carpeta de *Assets*, hasta el hueco de *AudioClip* del componente *AudioSource* será suficiente.

4.3. FUNCIONAMIENTO DEL TUTORIAL

4.3.1. INTRODUCCION

En este apartado se procederá a explicar cómo se ha creado y cómo funciona el tutorial y los diferentes elementos que lo conforman. También se explicará cómo interactúan y que finalidad tienen cada uno de los objetos presentes.

El orden en el que se realizará la explicación del programa será el mismo orden que se debería de seguir si se utilizase dicho programa de una forma correcta. Es decir, se irán explicando las cosas a medida que vayan sucediendo, de esta forma, también servirá como ayuda a la hora de utilizarlo.

También comentar que, en este apartado, se explicará el código utilizado para el funcionamiento del programa y para la interacción de los diferentes elementos.

4.3.2. PANTALLA INICIAL

El proyecto consta de dos escenas, una primera que es una pantalla inicial con diferentes opciones, y la segunda, que es donde sucede el tutorial. Por lo que, en cuanto se inicia la aplicación, lo primero que nos aparece es dicha pantalla inicial.

Para la creación de la escena, el primer paso ha sido colocar un plano a modo de fondo. También ha sido necesario crear dos *canvas* con sus respectivos textos y botones. La pantalla inicial funciona de la siguiente manera:

Primero aparece el primer *canvas* que contiene un texto y tres botones. En el texto aparece el título del programa, y los tres botones son el de empezar, el de ayuda y el de salir respectivamente.

Si se pulsa el botón de empezar, el programa saltará hacia la siguiente escena, es decir, a la del tutorial. Este botón, a diferencia del resto, ha necesitado un tipo de programación diferente, ya que Unity tiene una función propia para saltar a la siguiente escena.

Para ello, es necesario ir a la pantalla del inspector del botón, y en el componente "*Button (script)*" añadir una función. Las opciones a elegir son "*Runtime Only*", para solo realizar la acción cuando se solicite, y "*PantallaInicial.Inicio*" para llamar al código programado. Además, también hay que seleccionar el objeto donde se encuentra el script, que este caso es en el *Canvas1*. La forma resultante tiene que ser la siguiente:

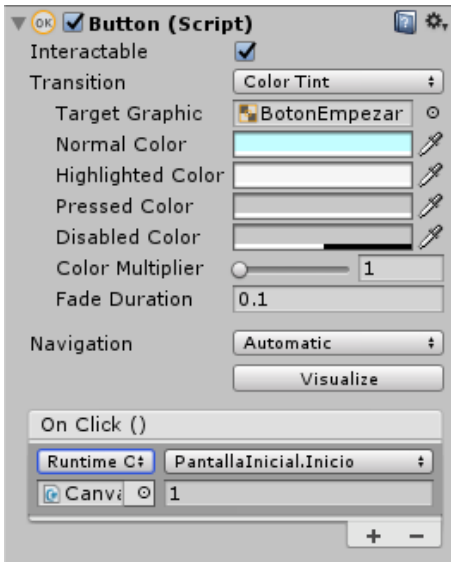

Figura 42. Componente para la configuración del botón.

El botón de ayuda en cambio nos hace aparecer una pantalla diferente, pero dentro de la misma escena. Esta nueva pantalla, que realmente es un *canvas* diferente, contiene un texto en el que aparece una breve descripción y explicación del programa, y un botón con el nombre volver. El botón de volver permite regresar a la pantalla anterior.

El último botón es el de salir. La función de este botón es la de cerrar la aplicación si se pulsa en él.

Figura 43. Pantalla inicial.

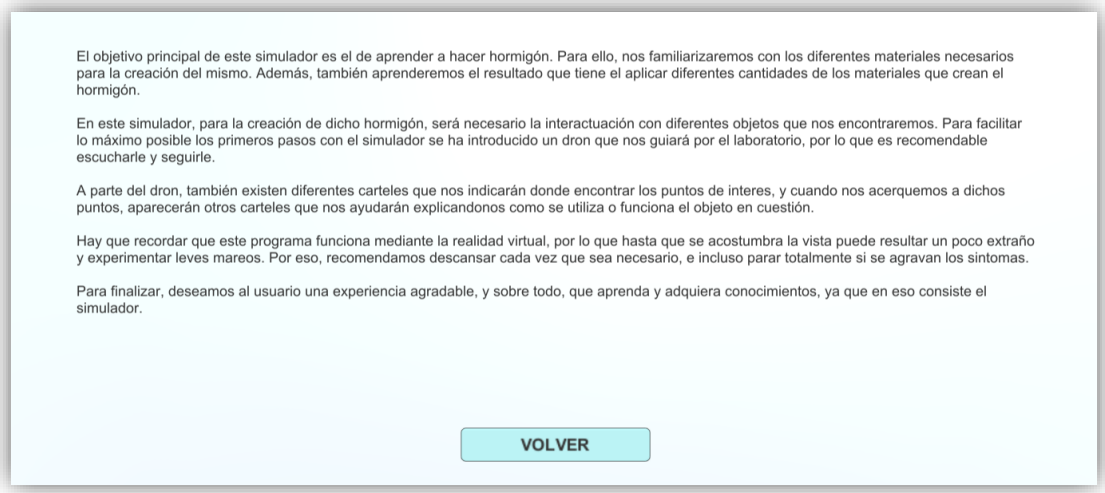

Figura 44. Pantalla de ayuda.

4.3.2.1. CODIGO UTILIZADO

Se va a fragmentar el código utilizado, para realizar una mejor explicación del mismo.

```
public Button BotonSalir;
public Button BotonAyuda;
public Button BotonVolver;

public GameObject Canvas1;
public GameObject Canvas2;
```

El primer paso es declarar los diferentes objetos que se utilizarán en la programación. Siempre que un objeto vaya a formar parte de la programación, es obligatorio declararlo con anterioridad. Para ello, se declaran los objetos como variables públicas, y luego, dentro de Unity, a dicha variable se le asigna el objeto correspondiente.

Figura 45. Controles para declarar los elementos.

```
public void Inicio(int scene)
{
    Application.LoadLevel(scene);
}
```

Mediante lo ya explicado en el apartado del botón Empezar, esta parte del código se activará cada vez que se haga clic en dicho botón. La función de este código es la de saltar de escena.

```
void Start () {
    BotonSalir.onClick.AddListener(TaskOnClick);
    BotonAyuda.onClick.AddListener(TaskOnClick1);
    BotonVolver.onClick.AddListener(TaskOnClick2);

    Canvas1.SetActive(true);
    Canvas2.SetActive(false);
}
```

Esta parte del código se ejecutará según arranque el programa. Se pueden diferenciar dos partes; una primera que es la de llamar la función correspondiente cada que vez se hace clic en un botón, y una segunda, que activa el Canvas1 y desactiva el Canvas2 cada vez que se inicializa la aplicación.

```
    void Update () {

}

void TaskOnClick()
{
    Debug.Log("You have clicked the button!");
    Application.Quit();
}
void TaskOnClick1()
{
    Debug.Log("You have clicked the button!");
    Canvas1.SetActive(false);
    Canvas2.SetActive(true);
}
void TaskOnClick2()
{
    Debug.Log("You have clicked the button!");
    Canvas1.SetActive(true);
    Canvas2.SetActive(false);
}
```

Estas son las funciones que se ejecutarán cada vez que sean llamadas, es decir, lo que sucederá cada vez que se pulse alguno de los botones. Cada vez que se llama la primera función, se consigue cerrar la aplicación. Con las otras dos en cambio, se consigue cambiar de pantalla, ya que se desactiva un *canvas* y se activa el otro.

4.3.3. TUTORIAL

Este apartado corresponde a la segunda escena del proyecto. La escena empezará en el momento en el que se pulsa el botón “Empezar”. Para explicar la escena, se seguirán los pasos ya establecidos.

4.3.3.1. MANEJO DEL PERSONAJE

Para el manejo del personaje, debido a que es necesario el uso del ratón para hacer clic en los diferentes botones, existen dos posibilidades diferentes.

La primera es teclado más ratón. Se utilizan las teclas W, A, S y D para movernos en las diferentes direcciones (W para ir hacia adelante, A para rotar a la izquierda, S para ir hacia atrás y D para rotar a la derecha) y la tecla F para activar los diferentes elementos.

La segunda opción es el mando más el ratón. El mando dispone de un *stick para* permitir el movimiento en cualquier dirección al igual que las teclas ya mencionadas. Además del *stick*, también dispone de un botón, el cual cumple la función de la tecla F.

La posibilidad de utilizar el ratón para seleccionar y no para mover la cámara, la da el dispositivo Oculus Rift gracias a su *Head Pose,* mediante el cual, al mover la cabeza, también moverá la cámara principal.

4.3.3.2. DRONE

Nada más iniciar la escena aparece un *drone* delante del personaje. La función de este *drone* es la de hacer de guía. Para cumplir con dicha función se le ha equipado con los componentes "*Audio Source*" y un "*Animation*", para que a la vez que se mueva vaya realizando las explicaciones.

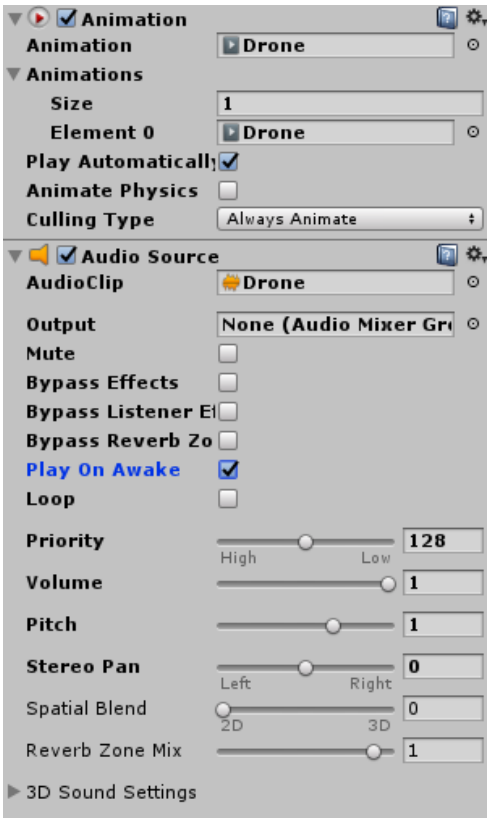

Figura 46. Componentes de animación y audio del drone.

Para conseguir con la mayor exactitud posible que el *drone* empiece y termine de hablar en el lugar correspondiente, se ha utilizado el ya mencionado programa de audio Audacity y el generador de animaciones del Unity. Los pasos a seguir han sido los siguientes:

- Se graba todo lo que el *drone* tiene que reproducir en cada lugar deseado en el programa Audacity.

- Se crea la animación para que el *drone* se mueva por todos los puntos de interés.

- Mientras se crea la animación, cada vez que se para el *drone* y antes de volver a moverse, hay que dejar un intervalo de tiempo suficiente para reproducir el trozo de audio correspondiente a ese lugar.

- Una vez se ha terminado la animación, en la ventana de *Animation* hay que observar cuanto tiempo tarda el *drone* de ir de un lugar al siguiente. Una vez se sabe esa cantidad de tiempo, en el Audacity hay que añadir intervalos de silencio iguales a dicha cantidad de tiempo.

Realmente el *drone* está animado y reproduciendo el audio todo el tiempo que dura la guía, pero en los momentos en los que está en movimiento el audio no tiene nada grabado, y los momentos en los que está quieto reproduciendo audio, la animación no tiene ningún movimiento programado.

Figura 47. Imagen resultante del drone en Unity.

4.3.3.3. PROFESOR

El profesor es un personaje no jugador cuya función es la de reproducir audios con información sobre los diferentes materiales que participan en el tutorial. Dichos audios también se han grabado con el programa Audacity, y una vez creados, se han importado al Unity.

Para seleccionar el audio deseado, se ha creado un *canvas* con diferentes botones, cada uno para reproducir el audio correspondiente. Además, el *canvas* no se activará si no se está a una distancia relativamente corta del profesor.

Figura 48. Imagen resultante del profesor en Unity.

En el momento en el que aparece la consola y se pulsa cualquiera de los botones, el componente *Audio Source* comienza a reproducir el audio deseado.

4.3.3.3.1. CODIGO UTILIZADO

Ahora se expondrá todo el código necesario para el funcionamiento de los botones.

```
public GameObject consola2;

public Button Bcemento;
public Button Baridos;
public Button Bagua;
public Button Baditivos;
```

El primer paso siempre es declarar las variables, en este caso, tenemos el *canvas* que es la consola2 y los cuatro botones. Como es una variable pública, en Unity a cada una de estas variables hay que asignarle un elemento, que lógicamente, será el *canvas* y los cuatro botones.

Figura 49. Elementos declarados.

```
    Bcemento.onClick.AddListener(TaskOnClick1);
    Baridos.onClick.AddListener(TaskOnClick2);
    Bagua.onClick.AddListener(TaskOnClick3);
    Baditivos.onClick.AddListener(TaskOnClick4);
```

Mediante esta parte del código, se consigue que cada vez que se pulse uno de los botones, llame a la correspondiente función. Por ejemplo, si se hace clic en el botón de Aridos, se llamaría a la función TaskOnClick2.

```
void TaskOnClick1()
{
    Debug.Log("Has pulsado el boton del cemento!");
    AudioSource audio = GetComponent<AudioSource>();
    audio.Play();
    audio.clip = Cemento;
    audio.Play(44100);
}
void TaskOnClick2()
{
    Debug.Log("Has pulsado el boton de los aridos!");
    AudioSource audio = GetComponent<AudioSource>();
    audio.Play();
    audio.clip = Aridos;
    audio.Play(44100);
}
void TaskOnClick3()
{
    Debug.Log("Has pulsado el boton del agua!");
    AudioSource audio = GetComponent<AudioSource>();
    audio.Play();
```

```
    audio.clip = Agua;
    audio.Play(44100);
}
void TaskOnClick4()
{
    Debug.Log("Has pulsado el boton de los aditivos!");
    AudioSource audio = GetComponent<AudioSource>();
    audio.Play();
    audio.clip = Aditivos;
    audio.Play(44100);
}
```

Estas son las funciones que se activarán al pulsar los botones. Como la función de los cuatro botones es similar, el código de las cuatro también será parecido. En concreto, solo cambia el clip de audio a reproducir, que para cada botón es uno diferente.

```
void Start()
{
    consola2.SetActive(false);

}

void OnTriggerEnter(Collider other)
{
    if (other.gameObject.name == "profesor")
    {
        consola2.SetActive(true);
        AyudaProfesor.SetActive(true);
    }

}

void OnTriggerExit(Collider other)
{
    if (other.gameObject.name == "profesor")
    {
        consola2.SetActive(false);
        AyudaProfesor.SetActive(false);
    }

}
```

Esta parte del código es la referente a la activación y desactivación de la consola. Aunque realmente en este caso el código no siga el mismo orden en Visual Studio, se ha colocado de esta manera para una mejor explicación.

La primera parte, la referente al *void Start*, sirve para que la consola empiece desactivada una vez se inicie el tutorial, ya que solo se quiere activada en un determinado momento.

Las otras dos partes están relacionadas, ya que la función es muy similar. Una es *OnTriggerEnter*, y la otra *OnTriggerExit*, es decir, la primera función se ejecutará cuando se entre en el *collider* correspondiente, y la segunda se ejecutará cuando se salga del *collider*. En este caso para ambas funciones el *collider* es el mismo, que es el "profesor".

La función que realiza cada una también es similar. En *OnTriggerEnter* se activará la consola, y en *OnTriggerExit* se desactivará. La última función también es de importancia, porque si no, cada vez que se entrase al *collider*, la consola se quedaría activada el resto del tutorial.

4.3.3.4. RECOLECTAR LOS MATERIALES

Para la creación del hormigón, son necesarios diferentes materiales, por lo que lo primero será recolectar dichos materiales. Para localizarlos basta con seguir al *drone*; además, encima de los mismos hay un cartel indicando la posición y el tipo de material en cuestión.

En el tutorial se pueden encontrar cuatro tipos de material diferentes: agua, cemento, áridos y aditivos. Cabe destacar que, dentro de los aditivos, existen tres subtipos diferentes, que son plastificantes, acelerantes y retardantes.

Figura 50. Diferentes materiales para recolectar.

La recolección de los cuatro materiales se realiza de la misma forma. Es necesario acercarse al material deseado hasta que se active el panel de ayuda, que significará que se está lo suficientemente cerca, y una vez activado, hay que pulsar la tecla “F” en el caso de usar el teclado, y el botón en el caso de usar el mando.

Hay que prestar especial atención cuando se quiera recoger cualquiera de los aditivos, ya que, al estar relativamente cerca, hay que ponerse exactamente delante del aditivo deseado antes de pulsar la tecla “F” o el botón.

4.3.3.4.1. CODIGO UTILIZADO

Se explicará el código utilizado para que sea posible la recolección de los materiales.

```
public int arido;
public int cemento;
public int agua;
public int retardante;
public int plastificante;
public int acelerante;
```

Al igual que en los anteriores casos, lo primero es declarar las variables. Se vuelven a hacer públicas porque luego los objetos se seleccionan desde el programa Unity.

```
inventory.Add("arido", arido);
inventory.Add("cemento", cemento);
inventory.Add("agua", agua);
inventory.Add("retardante", retardante);
inventory.Add("plastificante", plastificante);
inventory.Add("acelerante", acelerante);
```

Como se va a trabajar con diferentes objetos, para facilitar el código a utilizar, y que el programa trabaje lo más fluido posible, se crea un inventario para apilar los diferentes materiales.

```
void Update()
{

    foreach (var keyValue in inventory)
    {
        Debug.Log("Tengo " + keyValue.Value + " " + keyValue.Key);

    }
}
```

Aunque esta parte del código sirva para todos los objetos que se encuentren durante el tutorial, se procederá a explicar ahora. Este trozo de código se utiliza

para que cada vez que cambia el valor de algún objeto que se tenga en el inventario, aparezca en la consola y así confirmar que se ha obtenido.

Esto es de gran utilidad, ya que existen ciertos elementos que no funcionan si no se tienen los objetos requeridos, y gracias a este código, en todo momento se puede saber cuáles son los objetos que se han recolectado y cuáles no.

```
void OnTriggerStay(Collider other)
{
    bool fButton = Input.GetKeyDown(KeyCode.F);

    if (!fButton)
    {
        return;
    }

    if (other.gameObject.name == "cemento" && fButton)
    {
        Debug.Log("Estoy en el cemento.");
        inventory["cemento"] += 100;
    }

    if (other.gameObject.name == "agua" && fButton)
    {
        Debug.Log("Estoy en el agua.");
        inventory["agua"] = inventory["agua"] + 100;
    }

    if (other.gameObject.name == "arido" && fButton)
    {
        Debug.Log("Estoy en el arido.");
        inventory["arido"] = inventory["arido"] + 100;
    }

    if (other.gameObject.name == "acelerante" && fButton)
    {
        Debug.Log("Estoy en el acelerante.");
        inventory["acelerante"] = inventory["acelerante"] + 1;
    }
```

```
if (other.gameObject.name == "plastificante" && fButton)
{
    Debug.Log("Estoy en el plastificante.");
    inventory["plastificante"] = inventory["plastificante"] + 1;
}

if (other.gameObject.name == "retardante" && fButton)
{
    Debug.Log("Estoy en el retardante.");
    inventory["retardante"] = inventory["retardante"] + 1;
}
```

Lo que la primera línea de código indica, es que, para ejecutar la función, es necesario estar dentro del "*Collider*" correspondiente. Por lo que, en el programa, a todos los materiales a recolectar, se les ha añadido un componente de "*Collider*".

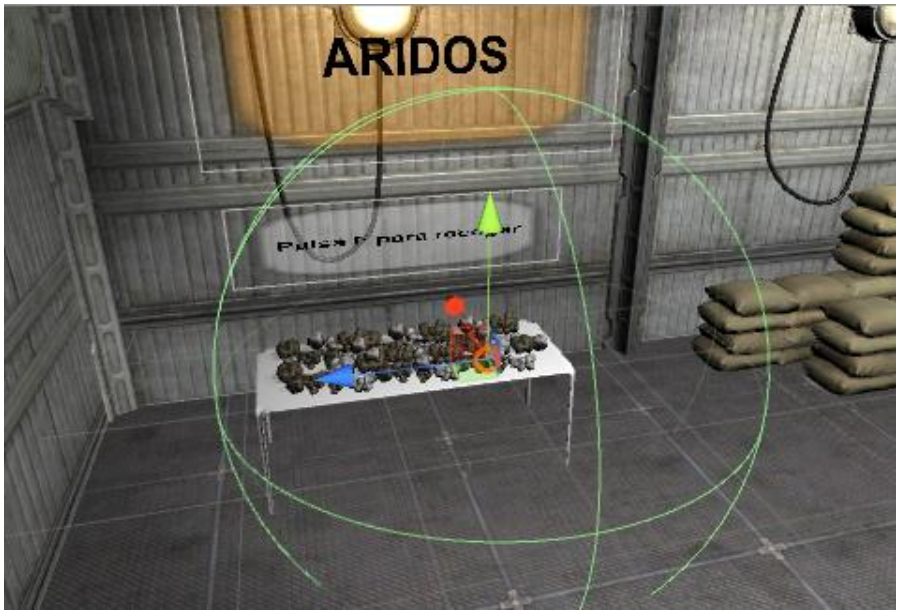

Figura 51. Collider del arido.

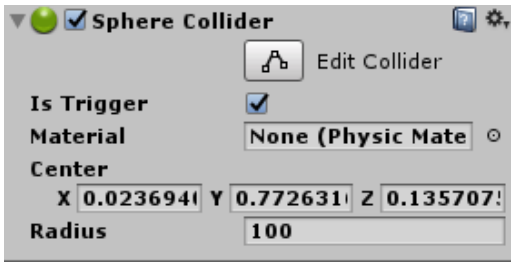

Figura 52. Caracteristicas del collider.

Fácilmente se puede apreciar, que el código está compuesto de diferentes "*If*", así que será necesario cumplir las condiciones establecidas para entrar dentro de dicho "*If*". Todos los "*If*" son similares y tienen una doble condición; la primera es estar dentro del *Collider* correspondiente, y la segunda es pulsar la tecla "F" o el botón.

Una vez satisfechas las condiciones del "*If*", se ejecuta su respectiva función, que en todas consiste en añadir al inventario el correspondiente material en la cantidad deseada.

4.3.3.5. MEZCLAR LOS MATERIALES

Una vez se obtienen todos los materiales deseados, el siguiente paso es mezclarlos para crear el hormigón. Dicho proceso se realiza en la hormigonera que se encuentra en el tutorial.

Figura 53. Hormigonera.

Este paso es el más importante de todos, ya que aquí se deciden las diferentes cantidades de material que se quieren utilizar; por lo que dependiendo de las decisiones que se tomen en este paso, se obtendrá un resultado final diferente.

Para facilitar que materiales usar y que cantidades de los mismos, al igual que con el profesor, se ha creado otra consola. A la consola se le han añadido diferentes *sliders* y *toggles* para que sean lo más intuitiva posibles las diferentes selecciones a realizar.

Una vez se finaliza la selección de materiales y cantidades, es necesario pulsar un botón que contiene también la consola. Este botón llamado Mezclar sirve para confirmar la elección realizada.

Figura 54. Consola para la mezcla de materiales.

Para explicar las diferentes acciones que realiza la hormigonera, y como usarla correctamente, se hará a medida que se vaya explicando el código, para poder hacer la explicación paso a paso.

4.3.3.5.1. CODIGO UTILIZADO

Este es el código necesario para un funcionamiento correcto:

```
public int objeto11;
public int objeto12;
```

```
public int objeto13;
public int objeto21;
public int objeto22;
public int objeto23;
public int objeto31;
public int objeto32;
public int objeto33;

public int probeta;

public GameObject consola;

public Button Mezclar;

public AudioClip Hormigonera;

public Slider SAgua;
public Slider SCemento;
public Slider SAmaximo;
public Slider SAminimo;

public Toggle TPlastificante;
public Toggle TAcelerante;
public Toggle TRetardante;
```

Al igual que en todos los casos anteriores, el primer paso aquí también será declarar todas las variables que aparecerán en esta parte del código. Y como son públicas, también será necesario asignarle los correspondientes objetos en el programa.

```
Mezclar.onClick.AddListener(TaskOnClick5);
```

Esta línea de código servirá para entrar en la función correspondiente cuando se pulse el botón Mezclar de la consola.

```
SAgua.onValueChanged.AddListener(UpdateAguaText);
SCemento.onValueChanged.AddListener(UpdateCementoText);
```

```
        SAmaximo.onValueChanged.AddListener(UpdateAmaximoText);
        SAminimo.onValueChanged.AddListener(UpdateAminimoText);


    private void UpdateAguaText(float arg0)
    {
        Text AguaText = SAgua.GetComponentInChildren<Text>();
        AguaText.text = "AGUA: " + SAgua.value + "%";
    }

    private void UpdateCementoText(float arg0)
    {
        Text CementoText = SCemento.GetComponentInChildren<Text>();
        CementoText.text = "CEMENTO: " + SCemento.value + "%";
    }

    private void UpdateAmaximoText(float arg0)
    {
        Text AmaximoText = SAmaximo.GetComponentInChildren<Text>();
        AmaximoText.text = "TAMAÑO MAX ARIDO: " + SAmaximo.value + "%";
    }

    private void UpdateAminimoText(float arg0)
    {
        Text AminimoText = SAminimo.GetComponentInChildren<Text>();
        AminimoText.text = "TAMAÑO MIN ARIDO: " + SAminimo.value + "%";
    }
```

En este trozo de código, la primera parte hace que se ejecute la función correspondiente cada vez que se produzca un evento, que, en este caso, es un *OnValueChanged*; es decir, cada vez que haya un cambio de valor en cualquiera de dichos elementos, el programa entrará en la función que le corresponda.

El resto de funciones, como se puede observar, son similares. Cada vez que se ejecuta una de esas funciones, que en este caso es cuando haya un cambio de valor, el texto pasará a ser el texto ya escrito más el nuevo valor de la *slider*.

Resumiendo, mediante este código, se sabrá en todo momento el valor que se le está asignando a la *slider* en cuestión.

```
  void TaskOnClick5()
    {
        Debug.Log(SAgua.value);
        Debug.Log(SCemento.value);
        float relacionAC = SAgua.value / SCemento.value;
        float relacionArido = SAmaximo.value / SAminimo.value;

        if (inventory ["agua"] !=0 && inventory["cemento"] !=0 && inventory
["arido"] != 0)
        {
            Debug.Log("Has pulsado el boton de mezclar!");
            AudioSource audio = GetComponent<AudioSource>();
            audio.Play();
            audio.clip = Hormigonera;
            audio.Play(44100);
            inventory["probeta"] = inventory["probeta"] + 1;


            if (inventory["agua"] != 0 && inventory["cemento"] != 0)
            {
                if (relacionAC < 0.35)
                {
                    Debug.Log(" objeto 1.1. ");
                    inventory["objeto11"] = inventory["objeto11"] + 1;
                    calidadMezcla = RelacionAC.FaltaAgua;
                }
                else if (relacionAC >= 0.35 && relacionAC < 0.42)
                {
                    Debug.Log(" objeto 1.2. ");
                    inventory["objeto12"] = inventory["objeto12"] + 1;
                    calidadMezcla = RelacionAC.Bien;
                }
                else
                {
                    Debug.Log(" objeto 1.3. ");
                    inventory["objeto13"] = inventory["objeto13"] + 1;
                    calidadMezcla = RelacionAC.Aguado;
                }
                inventory["agua"] = 0;
                inventory["cemento"] = 0;
            }
```

```
if (inventory["arido"] != 0)
{
    if (relacionArido < 1.4)
    {
        Debug.Log(" objeto 2.1.");
        inventory["objeto21"] = inventory["objeto21"] + 1;
    }

    else if (relacionArido <= 2 && relacionArido >= 1.4)
    {
        Debug.Log(" objeto 2.2.");
        inventory["objeto22"] = inventory["objeto22"] + 1;
    }

    else
    {
        Debug.Log(" objeto 2.3.");
        inventory["objeto23"] = inventory["objeto23"] + 1;
    }
    inventory["arido"] = 0;
}

if (TPlastificante.isOn && inventory["plastificante"] != 0)
{
    Debug.Log(" objeto 3.1.");
    inventory["objeto31"] = inventory["objeto31"] + 1;
    inventory["plastificante"] = 0;
}

if (TAcelerante.isOn && inventory["acelerante"] != 0)
{
    Debug.Log(" objeto 3.2.");
    inventory["objeto32"] = inventory["objeto32"] + 1;
    inventory["acelerante"] = 0;

}

if (TRetardante.isOn && inventory["retardante"] != 0)
{
    Debug.Log(" objeto 3.3.");
    inventory["objeto33"] = inventory["objeto33"] + 1;
    inventory["retardante"] = 0;
```

```
            }
        }

    }
```

Lo primero de todo, es tener claro que en esta función solo se entra si se pulsa el botón mezclar, es decir, todo lo que contiene esta función no se ejecutará hasta que se pulse dicho botón.

Las primeras acciones que realiza esta función son dos divisiones. Mediante la primera de ellas, se obtiene la relación agua cemento, que se consigue dividiendo el valor que se le da a la slider del agua, entre el valor que se le asigne a la *slider* del cemento.

En la segunda división, la diferencia que se consigue es la relación de los diferentes tamaños de áridos, y para ello, se divide el porcentaje de tamaño máximo del árido entre el porcentaje de tamaño mínimo del árido.

El siguiente paso es un "*If*", por lo que habrá que cumplir una condición para seguir adelante en la función. En este caso, la condición es tener agua, cemento y áridos; es decir, si en ese momento no se tiene cualquiera de los tres elementos, no se entrará en dicha función. La razón es que para hacer hormigón son necesarios esos elementos, por lo que no es lógico seguir adelante, si no se tienen todos.

Las siguientes líneas del código hacen que se active un clip de audio, y añade un objeto llamado "probeta" al inventario. Dicho objeto será necesario más adelante en el tutorial.

A continuación, se pueden observar diferentes "*If*" con diferentes opciones cada uno de ellos. Resumiendo, el programa, dependiendo de las diferentes cantidades que hayamos elegido, y de los aditivos que seleccionemos, entrará en un "*If*" u otro, y en todos en los que entre recibirá un objeto, que dependiendo de qué objeto sea, al final del programa se tendrá un resultado u otro.

Hay que tener claro, que para entrar en los "*If*", es necesario tener el elemento correspondiente. Por ejemplo, aunque se active la pestaña de retardante, si no

se ha recogido con anterioridad el objeto retardante, no se entrara en su "*If*", por lo que tampoco se obtendrá el objeto de dicho "*If*".

Cabe destacar que al final del código que se ejecuta cada vez que se entra en un "*If*", mediante la sentencia "inventory["objeto"] = 0", se borran los objetos del inventario; es decir, esto representa el haber usado un objeto, y haberlo gastado en dicho uso.

```
void Start()
{
    consola.SetActive(false);
}

void OnTriggerEnter(Collider other)
{
    if (other.gameObject.name == "HormigoneraCollider")
    {
        consola.SetActive(true);
        AyudaHormigonera.SetActive(true);
    }
}

void OnTriggerExit(Collider other)
{
    if (other.gameObject.name == "HormigoneraCollider")
    {
        consola.SetActive(false);
        AyudaHormigonera.SetActive(false);
    }
}
```

La explicación de esta parte del código es la misma que la hecha para explicar la consola del profesor. Al final, el funcionamiento de activación y desactivación de esta consola es idéntico a la otra, pero con la diferencia del *collider*. En el

anterior caso el *collider* era el “profesor”, pero en este caso lo es la “hormigonera”.

El método de funcionamiento es el siguiente: la consola empieza desactivada, al acercarse a la hormigonera la consola se activa, y al alejarse de la misma, vuelve a desactivarse.

4.3.3.6. PRENSA

Después de mezclar los materiales y cantidades deseados, se obtiene la probeta en cuestión, suponiendo que fragua al instante. Para saber exactamente las características que tiene, será necesario poner la probeta en una prensa, y mediante un ensayo, enviará el resultado obtenido al ordenador.

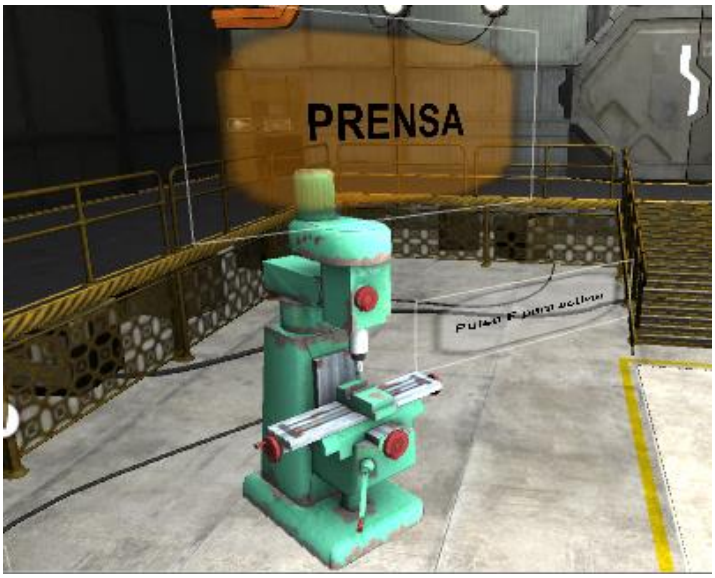

Figura 55. Prensa.

4.3.3.6.1. CODIGO UTILIZADO

A continuación, se explicará el código utilizado para el correcto funcionamiento de la prensa.

```
public AudioClip Prensa;
```

Como la prensa utiliza un clip de audio, al igual que el resto de los objetos, es necesario declararlo primero. Una vez declarado, en el programa Unity se le asignara el clip de audio correspondiente.

```
        if (other.gameObject.name == "Prensa" && fButton)
        {
            Debug.Log("Estoy en la prensa.");

            if (other.gameObject.GetComponent<Animation>().enabled == false &&
inventory["probeta"] != 0)
                {
                    AudioSource audio = GetComponent<AudioSource>();
                    audio.Play();
                    audio.clip = Prensa;
                    audio.Play(44100);
                    other.gameObject.GetComponent<Animation>().enabled = true;
                    StartCoroutine(StopPrensa(other.gameObject));
                }
        }
```

Lo primero, comentar que este código está dentro de la función *OnTriggerStay*, por lo que nada de este código funcionará si no se está dentro del *collider* correspondiente; en este caso, dentro del *collider* de la “prensa”.

La primera línea del código es un “*If*”, lo que significa que habrá que cumplir alguna condición para entrar dentro. En este caso es una condición doble; la primera es la ya comentada estar dentro del *collider* de la “prensa”, y la segunda es pulsar el botón “F” o el botón del mando. Para entrar, es necesario cumplir ambas.

A continuación, encontramos otro “*If*” de condición doble también. La primera condición es que la animación de la prensa tiene que estar desactivada. La razón es que el código dentro del “*If*” inicializa una animación, así que para que no se solapen y de error, no se podrá seguir adelante hasta que la animación esté terminada por completo.

La segunda condición es tener en el inventario el objeto probeta; como es lógico, la prensa no debe activarse si no hay probeta alguna que ensayar.

Dentro del "*If*" ocurren tres cosas. La primera es activar un audio, la segunda es activar la animación, y la tercera es ejecutar la siguiente función:

```
IEnumerator StopPrensa(GameObject prensa)
{
    yield return new WaitForSeconds(11);
    prensa.GetComponent<Animation>().enabled = false;
}
```

Esta función se utiliza porque a las animaciones en Unity, si no se les da una cierta duración al crearlas, lo normal es que se pongan en bucle y no se finalicen nunca. Pero mediante este código, aunque la animación entre en dicho bucle, se para transcurrido un determinado tiempo; en este caso 11 segundos.

4.3.3.7. PROYECTOR

El último paso a realizar es conocer las características finales del hormigón creado. Para ello, la prensa manda la información obtenida al ordenador, y una vez dicha información está en el ordenador, puede ser reproducida mediante el proyector para que sea fácilmente visible.

Hay que tener en cuenta, que el resultado obtenido es reproducido por el proyector, por lo que está reproducción aparece en frente del proyector; que se encuentra a mano derecha de nuestra posición.

Figura 56. Proyector.

Claro está, que, para realizar este último paso, es necesario haber realizado de forma correcta todos los pasos anteriores.

4.3.3.7.1. CODIGO UTILIZADO

Este es el código utilizado para la reproducción de los resultados.

```
public GameObject LuzP;
public GameObject PanelResultado;
public GameObject TextoResultado;
```

El primer paso es declarar las variables que formarán parte del resto del código, y asignarles sus respectivos elementos del programa.

```
LuzP.SetActive(false);

PanelResultado.SetActive(false);
```

Estas dos líneas del código se encuentran dentro de la función *Start*, por lo que se ejecutarán nada más empezar el programa. La función de ambas es la misma, y es que empiecen las dos desactivadas, tanto la luz del proyector, como el panel reproducido por el proyector.

```
        if (other.gameObject.name == "TextoResultado" && inventory["probeta"] != 0 && fButton)
        {
            Debug.Log("Estoy en el Resultado.");
            string text = "Resultado obtenido: \n";
                if (inventory["objeto11"] > 0)
                {
                    text = text + "La relación agua/cemento es demasiado baja, por lo que el hormigón no va a hidratarse adecuadamente. Es necesario utilizar más agua.\n";
                    inventory["objeto11"] = 0;
                }
                if (inventory["objeto12"] > 0)
                {
                    text = text + "La relación agua/cemento es adecuada. La resitencia que alcanzará el hormigón será la óptima. Es probable que la trabajabilidad del hormigón sea difícil.\n";
                    inventory["objeto12"] = 0;
                }
                if (inventory["objeto13"] > 0)
                {
                    text = text + "El hormigón tiene demasiada agua, por lo que la resistencia será inferior a la óptima. Se obtendrá un hormigón muy trabajable.\n";
                    inventory["objeto13"] = 0;
                }
                if (inventory["objeto21"] > 0)
                {
                    text = text + "La granulometría no es la correcta. Esto puede deberse a dos razones; el tamaño máximo del arido no es lo suficientemente grande o el tamaño mínimo del arido es demasiado grande.\n";
                    inventory["objeto21"] = 0;
                }
                if (inventory["objeto22"] > 0)
                {
                    text = text + "Es correcta la granulometría. Los áridos más pequeños podrán rellenar los huecos que se crean entre los aridos de mayor tamaño.\n";
                    inventory["objeto22"] = 0;
                }
```

```
                if (inventory["objeto23"] > 0)
                {
                    text = text + "El tamaño medio del árido es demasiado grande. Los huecos entre los áridos más grandes no podran rellenarse mediante otros áridos.\n";
                    inventory["objeto23"] = 0;
                }
                if (inventory["objeto31"] > 0)
                {
                    text = text + "Mediante el plastificante utilizado se obtendrá una mejor trabajabilidad del hormigón sin influir en su resistencia.\n";
                    inventory["objeto31"] = 0;
                }
                if (inventory["objeto32"] > 0)
                {
                    text = text + "Con el uso del acelerante se obtendrá una mayor rapidez en el fraguado del hormigón, obteniendo un endurecimiento más rapido del mismo. \n";
                    inventory["objeto32"] = 0;
                }
                if (inventory["objeto33"] > 0)
                {
                    text = text + "Con el uso del retardante el fraguado del hormigón será más lento, es decir, tardará más en endurecer.\n";
                    inventory["objeto33"] = 0;
                }

                other.gameObject.GetComponent<Text>().enabled = true;
                other.gameObject.GetComponent<Text>().text = text;

                LuzP.SetActive(true);
                PanelResultado.SetActive(true);

                StartCoroutine(HideResultado(other.gameObject));

                inventory["probeta"] = 0;
        }
    }
```

Esta parte del código puede separarse en dos partes, una primera que estaría formada por todos los "*If*", y las últimas seis líneas del código; por lo que la explicación se procederá a hacer teniendo en cuenta estas dos partes.

La primera parte es la que decidirá que partes del texto serán incluidas en el resultado final y cuáles no. Para tomar dicha decisión, se realiza mediante los "*If*", ya que estos aplican condiciones. Por lo que para determinar que parte del texto estará incluida y cual no, dependerá de si cumple la condición o no.

Todas las condiciones de los "*If*" son similares. Se entrará o no en un determinado "*If*", si se tiene o no el correspondiente objeto. Dichos objetos se consiguen en el paso de la hormigonera, dependiendo de los materiales y cantidades utilizadas.

Resumiendo, dependiendo de los materiales y las cantidades utilizadas, la hormigonera dará unos u otros objetos; y entonces, el proyector solo pondrá el texto relativo a los objetos conseguidos.

Por eso, a diferente cantidad o diferentes materiales usados, diferente texto se obtendrá en el proyector.

En la segunda parte encontramos tres funciones diferentes. La primera son las cuatro primeras líneas, que activan el texto, el panel que lo contiene, y la luz del proyector.

La segunda parte, llama a la siguiente función:

```
IEnumerator HideResultado(GameObject TextoResultado)
{
    yield return new WaitForSeconds(20);
    TextoResultado.GetComponent<Text>().text = "";
}
```

Esta función lo que se hace es borrar el texto del resultado obtenido una vez hayan pasado veinte segundos. De esta manera, si se vuelve a probar con otros materiales y otras cantidades, puede aparecer un nuevo texto.

La tercera y última parte, elimina la probeta del inventario, para poder realizar una nueva con diferentes características.

La idea es que después de haber realizado un tipo de hormigón, se pueda volver a realizar un segundo sin tener que reiniciar el programa. Por eso, se busca en la medida de lo posible, eliminar todos los objetos que se vayan recolectando durante el tutorial, y eliminar el texto final.

4.3.3.8. PANELES DE AYUDA

Para facilitar en el mayor grado posible el transcurso del tutorial, en todos los puntos de interés se han colocado diferentes paneles de ayuda. Estos paneles ofrecen información; por ejemplo, de cómo utilizar el punto de interés en cuestión.

En algunos casos, al tratarse de paneles de un tamaño considerable, se ha optado por activarlos solo en el momento en que se acerque a los puntos de interés, es decir, el resto del tiempo estarán desactivados, para no romper la estética del entorno.

Estos paneles están formados por un *canvas* que contiene un panel y dentro del mismo un texto. Lo único que varía de un panel a otro es el texto escrito.

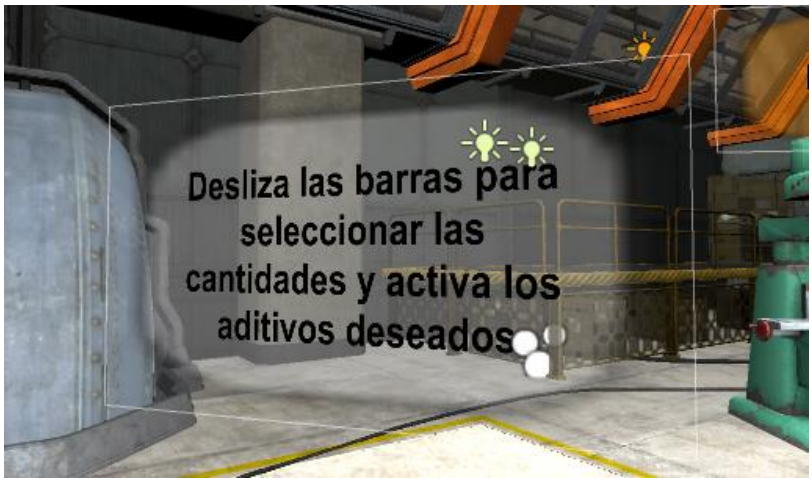

Figura 57. Paneles de ayuda.

4.3.3.8.1. CODIGO UTILIZADO

Este es el código utilizado para activar los carteles solo en el momento deseado.

```
void OnTriggerEnter(Collider other)
{

    if (other.gameObject.name == "HormigoneraCollider")
    {
        consola.SetActive(true);
        AyudaHormigonera.SetActive(true);
    }
    if (other.gameObject.name == "profesor")
    {
        consola2.SetActive(true);
        AyudaProfesor.SetActive(true);
    }
    if (other.gameObject.name == "arido")
    {
        AyudaRecoger.SetActive(true);
    }
    if (other.gameObject.name == "cemento")
    {
        AyudaRecoger2.SetActive(true);
    }
    if (other.gameObject.name == "agua")
    {
        AyudaRecoger3.SetActive(true);
    }
    if (other.gameObject.name == "retardante")
    {
        AyudaRecoger4.SetActive(true);
        AyudaRetardante.SetActive(true);
    }
    if (other.gameObject.name == "plastificante")
    {
        AyudaRecoger4.SetActive(true);
        AyudaPlastificante.SetActive(true);
    }
    if (other.gameObject.name == "acelerante")
    {
```

```
            AyudaRecoger4.SetActive(true);
            AyudaAcelerante.SetActive(true);
        }
        if (other.gameObject.name == "Prensa")
        {
            AyudaActivar.SetActive(true);
        }
        if (other.gameObject.name == "TextoResultado")
        {
            AyudaActivar2.SetActive(true);
        }
    }
```

Al tratarse de una función disparada por el evento *OnTriggerEnter*, solo se activará al entrar en el *collider* correspondiente. Una vez el programa detecta que se ha entrado, se activará el panel correspondiente.

```
    void OnTriggerExit(Collider other)
    {
        if (other.gameObject.name == "HormigoneraCollider")
        {
            consola.SetActive(false);
            AyudaHormigonera.SetActive(false);
        }
        if (other.gameObject.name == "profesor")
        {
            consola2.SetActive(false);
            AyudaProfesor.SetActive(false);
        }
        if (other.gameObject.name == "arido")
        {
            AyudaRecoger.SetActive(false);
        }
        if (other.gameObject.name == "cemento")
        {
            AyudaRecoger2.SetActive(false);
        }
        if (other.gameObject.name == "agua")
        {
```

```
            AyudaRecoger3.SetActive(false);
        }
        if (other.gameObject.name == "retardante")
        {
            AyudaRecoger3.SetActive(false);
            AyudaRetardante.SetActive(false);
        }
        if (other.gameObject.name == "plastificante")
        {
            AyudaRecoger3.SetActive(false);
            AyudaPlastificante.SetActive(false);
        }
        if (other.gameObject.name == "acelerante")
        {
            AyudaRecoger3.SetActive(false);
            AyudaAcelerante.SetActive(false);
        }
        if (other.gameObject.name == "Prensa")
        {
            AyudaActivar.SetActive(false);
        }
        if (other.gameObject.name == "TextoResultado")
        {
            AyudaActivar2.SetActive(false);
        }
    }
```

Una vez se entra al *collider* y se activa el panel, es necesario desactivarlo cuando se sale de dicho *collider*. Para ello se utiliza la función *OnTriggerExit*, que se ejecutará al salir. Cuando se ejecuta la función, el código escrito hace que se desactive el panel.

4.3.3.9. INTERFAZ DEL INVENTARIO

Además de los paneles, también se ha añadido otra ayuda para el inventario, para saber en todo momento de que materiales se dispone y de cuáles no. Esta interfaz puede apreciarse en todo momento en la parte superior izquierda de la pantalla.

Para la creación de la misma, se ha utilizado un *canvas* en el cual se han añadido diferentes textos. Se ha dispuesto de más de un texto debido a que hay momentos en los que solo es necesario cambiar uno de ellos, y no todos.

Figura 58. Inventario.

4.3.3.9.1. CODIGO UTILIZADO

El código empleado es el siguiente:

```
public GameObject Inventario;
public GameObject InventarioC;
public GameObject InventarioAr;
public GameObject InventarioAg;
public GameObject InventarioAd;
```

Lo primero de todo es declarar los diferentes objetos que se utilizarán.

```
void Update()
{

    {
        Debug.Log("Tengo " + keyValue.Value + " " + keyValue.Key);

    }
```

```
        if (inventory["cemento"] != 0)
        {
            InventarioC.GetComponent<Text>().text = "Cemento: SI";
        }
        else
        {
            InventarioC.GetComponent<Text>().text = "Cemento: NO";
        }

        if (inventory["agua"] != 0)
        {
            InventarioAg.GetComponent<Text>().text = "Agua: SI";
        }
        else
        {
            InventarioAg.GetComponent<Text>().text = "Agua: NO";
        }

        if (inventory["arido"] != 0)
        {
            InventarioAr.GetComponent<Text>().text = "Arido: SI";
        }
        else
        {
            InventarioAr.GetComponent<Text>().text = "Arido: NO";
        }

        if (inventory["plastificante"] == 0 && inventory["retardante"] == 0 &&
inventory["acelerante"] == 0)
        {
            InventarioAd.GetComponent<Text>().text = "Aditivo: NO";
        }
        else
        {
            InventarioAd.GetComponent<Text>().text = "Aditivo: SI";
        }
    }
```

Es importante darse cuenta de que el código empleado se encuentra dentro de un *Void Update*, es decir, este trozo de código se estará ejecutando en todo momento debido a la necesidad de saber en cada momento de que materiales se dispone y de cuáles no.

El código se basa en diferentes "*If*" que se activarán o no, dependiendo de si se tiene el material en cuestión o no. En el caso de que se tenga el material, el texto se modificará para que ponga "SI", y en el caso de que no se tenga, el texto se modificará para que ponga "NO".

4.4. CINETOSIS

El uso de aparatos de realidad virtual puede generar mareos en el usuario. Esto se debe al conflicto que crean las señales sensoriales en el cerebro, ya que se crea una discordancia entre la sensación visual de la velocidad y la sensación de movimiento del oído interno.

Explicado de una manera más sencilla, el mareo se debe a que cuando una persona mueve la cabeza, su cerebro espera que el entorno cambia exactamente en sincronía, por lo que, si se genera algún retraso perceptible, éste puede llegar a crear una sensación de incomodidad.

Además, aunque los usuarios no sean propensos a sufrir mareos en la vida real, los aparatos de realidad virtual generan una experiencia de inmersión tan grande, que igualmente puede generar mareos.

Las siguientes estrategias pueden prevenir y ayudar a disminuir la sensación de mareo:

- Leer toda la información de seguridad y salud incluida en los aparatos de realidad virtual para detectar posibles incompatibilidades.

- No forzar la situación. En el momento que se detecte algún síntoma de nausea es mejor parar e intentarlo más adelante, sin forzar en ningún momento el cuerpo.

- Detectar lo más rápido posible el síntoma de mareo. Antes de llegar a un estado de mareo es probable que se sufra otro tipo de síntoma, como por ejemplo dolor de cabeza; por lo que es recomendable parar en cuanto se detecte ese primer síntoma y no esperar a llegar al estado de mareo total.

- Ajustar correctamente el aparato de realidad virtual a la cabeza. Como, por ejemplo, que la correa no apriete la cabeza, o que la distancia entre los ojos y las lentes sea la correcta.

- Utilizar el perfil correcto de usuario. Algunos aparatos de realidad virtual permiten crear diferentes perfiles con las correspondientes características físicas, por lo que una vez creado dicho perfil, es recomendable utilizarle siempre.

- Empezar poco a poco en el uso. Esto se refiere a dos cosas. La primera es empezar con intervalos de tiempo cortos, y la segunda, empezar con aplicaciones ya prefabricadas para la pre familiarización de la realidad virtual.

- Estar sentado en el uso de la realidad virtual. Aunque estar de pies genere una sensación de inmersión mayor, el estar sentado mientras se usa genera una menor sensación de mareo.

- Reducir el nivel de brillo. A mayor nivel de brillo mayor será el contraste generado, y, por consiguiente, mayor será la probabilidad de tener una sensación de mareo.

- Utilizarlo en un espacio donde haya silencio. Cuantas más interferencias provengan del exterior, mayor discordancia va a detectar el cerebro, por lo que mayor sensación de mareo se va a experimentar.

- Tomar todos los descansos necesarios, y que sean del tiempo requerido para recuperar completamente.

4.5. CREAR LA APLICACIÓN

El paso final para la conclusión del trabajo es crear una aplicación del proyecto realizado. Explicado de una forma sencilla, la aplicación lo que hace es ejecutar directamente el trabajo, como si entrases en Unity y pulsases el botón de “Play”.

Para crear una aplicación de un proyecto con más de una escena es necesario tener una concordancia entre dichas escenas. Es decir, dentro del programa tenemos que tener disponible alguna forma de saltar entre las escenas.

Los pasos a seguir en Unity son los siguientes. En la barra de herramientas se pulsa el menú *File*, y dentro se hace clic en *Build Settings*. Esto abrirá la siguiente ventana.

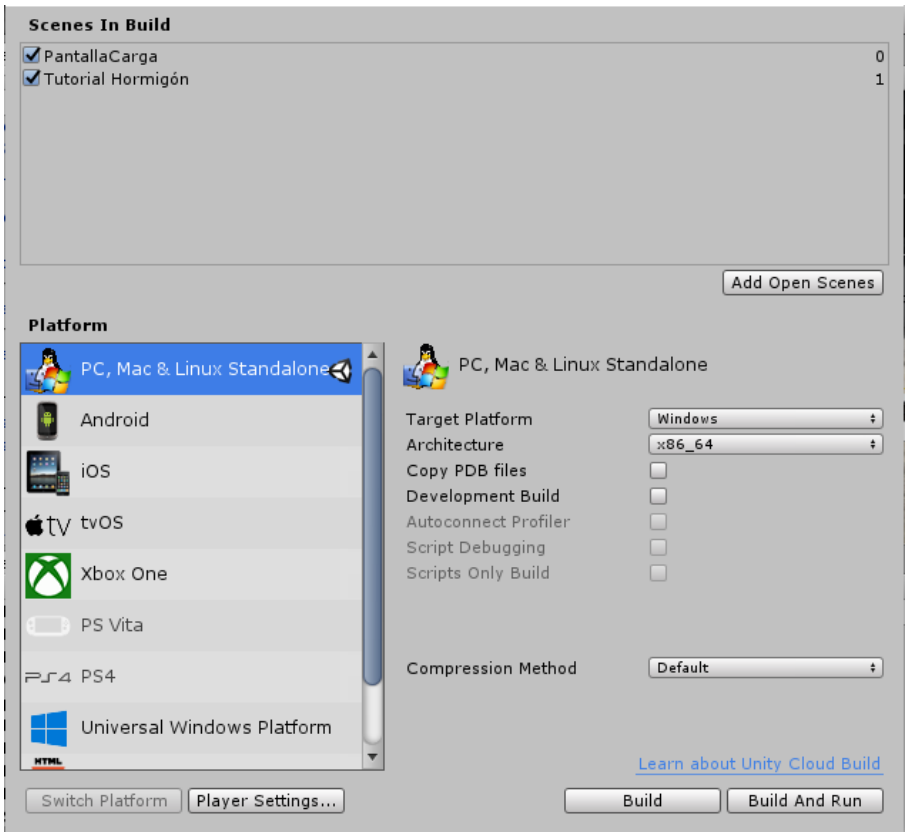

Figura 59. Ventana para la creación de la aplicación.

En esta ventana hay tres pasos a seguir de gran importancia.

El primero es seleccionar las escenas que se quieren incluir en la aplicación y el orden de las mismas. Esto es muy importante, porque seguramente la aplicación creada tendrá unos saltos de escena con un cierto orden. Por lo que, si no se colocan de una manera determinada, no se podrá seguir el orden establecido.

La segunda es seleccionar la plataforma deseada. Unity ofrece una gran variedad de plataformas para las que crear las aplicaciones.

La tercera y última es la pestaña de *Player Settings*. En esta pestaña se podrán modificar todas las características deseadas de la aplicación. En este caso tiene mayor importancia todavía, ya que es necesario que este activada la pestaña de la realidad virtual, y añadidos los tipos de dispositivos que se van a utilizar.

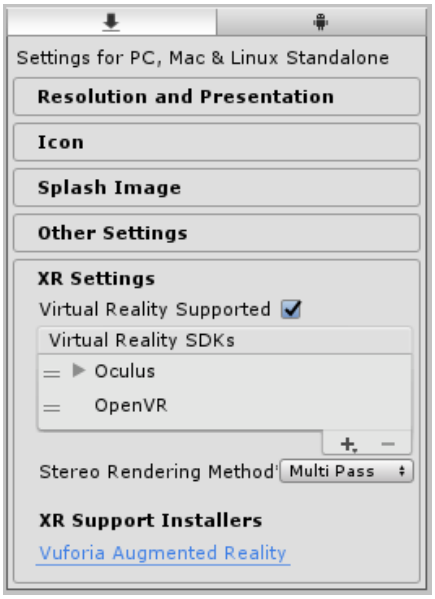

Figura 60. Ventana para la configuración de la realidad virtual.

Una vez creada la aplicación, la ventaja que ofrece, es que a partir de este momento no importa con que programa este hecha la aplicación, si no para que plataforma este generada. Por ejemplo, si se crea para un sistema operativo de escritorio, aunque la aplicación está desarrollada en Unity, luego no importara que dicho ordenador tenga o no instalado Unity, ya que dicha aplicación la ejecutará el sistema operativo.

5. CONCLUSIONES

- El haber optado por Unity como programa para la construcción del entorno en 3D ha ayudado en gran medida en la realización del trabajo gracias a su interfaz intuitiva y su facilidad para importar modelos 3D. Otros programas, como puede ser el Unreal Engine, son programas con un grado de dificultad más elevados.

- En relación con lo anterior, Unity también ha ofrecido la posibilidad de trabajar con el lenguaje de programación C#, el cual destaca por su sencillez. Además, el propio Unity cuenta con funciones propias que han aligerado el trabajo de programación.

- Gracias a la tecnología 3D se garantiza un mejor aprendizaje debido a que dicha tecnología crea en el usuario un mayor grado de atención y, por consiguiente, una mejor asimilación y retención de la información. Además, los entornos tridimensionales también ofrecen una mayor apreciación de los elementos con los que se interactúa.

- También se ha encontrado realmente útil la importación de los diferentes audios en el programa, los cuales, introducidos en diferentes elementos, como por ejemplo en el dron o en el profesor, ayudan en la realización del tutorial.

- Por otra parte, la utilización conjunta del programa Unity con el dispositivo de realidad virtual Oculus Rift ofrece una configuración automática del dispositivo, por lo que crea una gran ayuda en la utilización y en la calibración del mismo.

- Por último, debido al auge esperado en la tecnología de la realidad virtual, destacar la utilidad del trabajo como posible base para iniciarse en dicha tecnología y en el programa informático Unity 3D.

6. BIBLIOGRAFIA

BIBLIOGRAFÍA

Dr. Edward Lavieri. 2015. *Getting Started with Unity 5.* Birmingham : Packt Publishing Ltd., 2015. ISBN 978-1-78439-831-6.

Bradley Austin Davis, Karen Bryla, Philips Alexander Benton. 2015. *Oculus Rift in Action.* Shelter island : Manning Publications Co., 2015. ISBN 9781617292194

Simon Jackson. 2015. *Unity 3D UI Essentials.* Birmingham : Packt Publishing Ltd., 2015. ISBN 978-1-78355-361-7.

Linowes, Jonathan. 2015. *Unity Virtual Reality Projects.* Birmingham : Packt Publishing Ldt., 2015. ISBN 978-1-78398-855-6.

Nicolás Arrioja Landa Cosio. 2010. *C# Guía total del programador.* Buenos Aires : Fox Andina, 2010. ISBN 978-987-26013-5-5.