



ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

APLICACIÓN CON LENGUAJE JAVASCRIPT Y C# DE
ENTRETENIMIENTO BASADO EN UNITY 3D

Asier Echeverria Esparza

Oscar Ardaiz

Pamplona, 15 de Junio de 2013



Índice:

| | |
|--|-----|
| 1. Resumen..... | 3 |
| 2. Introducción | 4 |
| 2.1. Antecedentes..... | 4 |
| 2.2. Objetivos..... | 27 |
| 2.3. Fases del proyecto..... | 27 |
| 2.4. Planificación..... | 28 |
| 3. Desarrollo | 29 |
| 3.1. Formación en Unity 3D..... | 29 |
| 3.2. Compilación y ejecución en android..... | 62 |
| 3.3. Análisis..... | 66 |
| 3.4. Diseño e implementación..... | 69 |
| 3.6. Problemas encontrados | 127 |
| 4. Conclusiones y líneas futuras..... | 128 |
| 4.1. Conclusiones técnicas..... | 128 |
| 4.2. Conclusiones personales | 129 |
| 4.3. Líneas futuras | 130 |
| 5. Bibliografía..... | 131 |



1. Resumen:

En este documento se recoge la memoria del Proyecto de Fin de Carrera para la obtención del título de Ingeniero Técnico Informático de Gestión en la Universidad Pública de Navarra.

Este proyecto que lleva por título "Aplicación con lenguaje JavaScript y C# de Entretenimiento basado en Unity 3D" ha sido realizado por Asier Echeverria Esparza.

El encargado de supervisar el desarrollo del proyecto ha sido Oscar Ardaiz, tutor y profesor en la Universidad Pública de Navarra.

En general, el proyecto consiste en una aplicación de entretenimiento destinada para el uso en móviles que dispongan del sistema operativo Android, con el cual, el usuario podrá interactuar con la aplicación para conseguir una serie de objetivos. Con esta aplicación se intenta que el usuario pase un rato agradable, además, ha sido construido de manera que su uso sea lo más cómodo e intuitivo.

Para la realización de este proyecto, se han seguido todas las fases del ciclo de vida en un producto: especificación de requisitos, análisis, diseño, implementación y pruebas.



2. Introducción:

2.1. Antecedentes:

Nuestro deseo era construir una aplicación para Android de entretenimiento, es decir, un juego, por lo que tuvimos que informarnos cual eran los procesos para conseguir diseñar, implementar y que funcionase una aplicación sobre un móvil con sistema operativo Android.



Para ello, ahora vamos a explicar tres aspectos necesarios para la construcción de una aplicación en Unity 3D para Android:

- Android SDK
- Unity 3D
- JavaScript
- C Sharp



Android SDK:

Android SDK es el kit de desarrollo de software con el cual podemos desarrollar aplicaciones para dispositivos que dispongan del sistema operativo Android. Las plataformas que soporta son Linux (cualquier distribución moderna), Mac OS X 10.4.9 o superior y Windows XP o posterior.

Las aplicaciones pueden ser vendidas a través del Android Market y los desarrolladores que publiquen sus aplicaciones en este market reciben el 70% de los ingresos por ventas y Google se queda con el 30% restante. Los pagos los realiza Google a través del Google checkout. Por lo general el número de aplicaciones gratuitas es mucho mayor que el de las de pago, y realizan servicios similares.





Formación básica en Unity 3D

Unity 3D es una herramienta de programación integrada para la creación de video, juegos 3D u otros contenidos interactivos como visualizaciones arquitectónicas o animaciones 3D en tiempo real. Unity permite crear juegos para las siguientes plataformas: Windows, Mac OS X, Wii, iPhone/iPad, Xbox 360, Android y Play Station 3.

El desarrollador de este software es Unity Technologies, una empresa con sede en San Francisco y varias oficinas de desarrollo en Dinamarca, Lituania y Reino Unido. Unity Technologies está revolucionando la industria del juego con Unity 3d por su plataforma de desarrollo de gran avance para la creación de juegos 3D, aplicaciones interactivas en 3D, por simulaciones y visualizaciones de formación médica y arquitectónica, en la web, el IOS, Android, y más consolas (Wii y Play Station 3).

Los fundadores de Unity Technologies son David Helgason, Nicholas Francis y Joachim Arte. Utilizan un modelo de negocio que ha revolucionado el modelo de negocios de los juegos. Es gratis para una gran proporción de desarrolladores y asequibles para el resto, con unos ingresos que son los suficientemente fuertes para mantener Unity Technologies como un negocio rentable y que crece rápidamente. Es una empresa privada con el 50 % de los ingresos fuera de EEUU, de los que el 30 % de los ingresos no están relacionados con los juegos.

Sus productos son: Unity 3D, Unity Pro, Asset Server, iOS e iOS Pro.

- **Interfaz Grafica:**

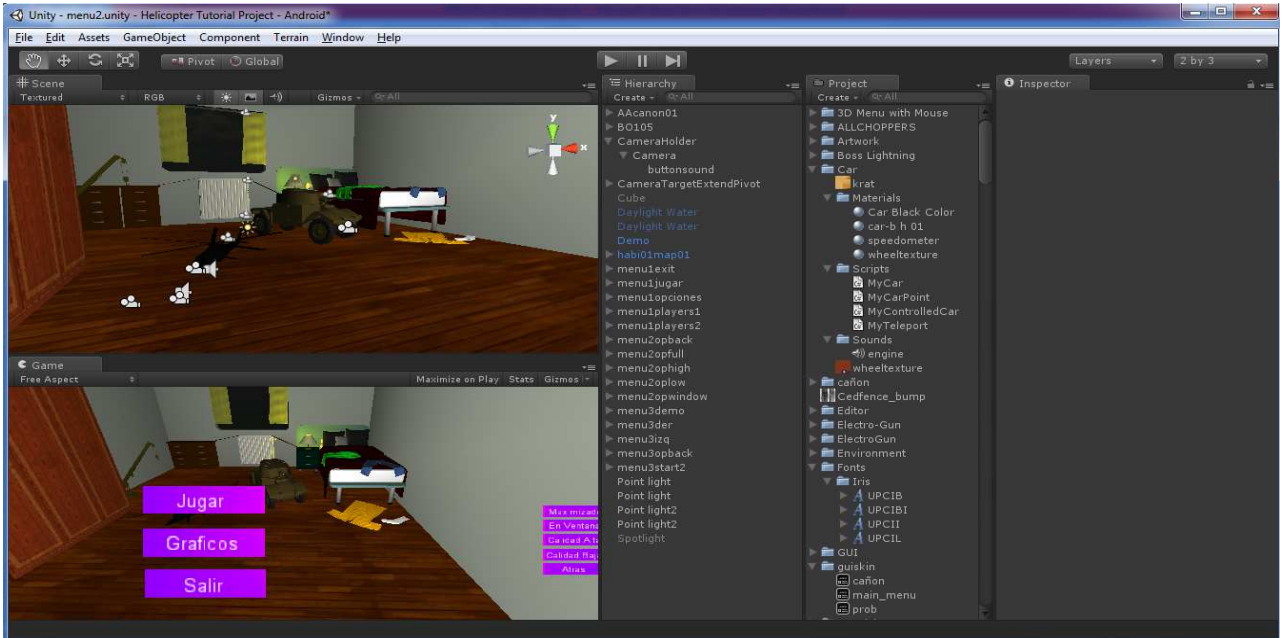
En este apartado se explicaran la interfaz gráfica para realizar operaciones básicas y las funciones principales para navegar a través de las secciones que Unity ofrece.

- **Main Editor Window:**

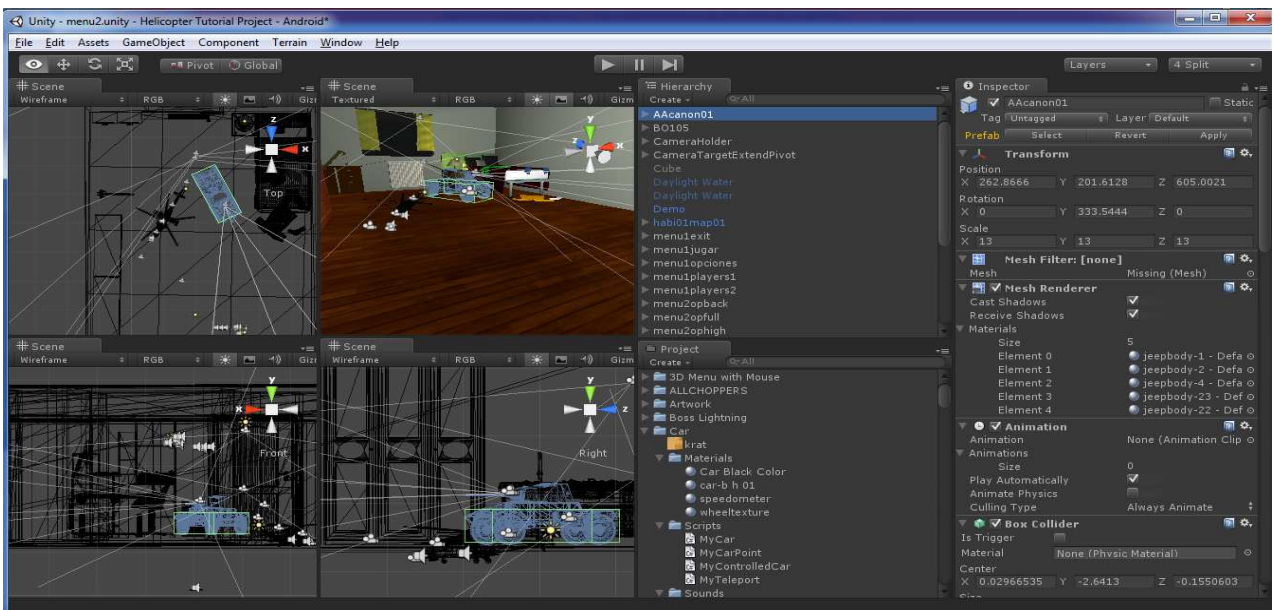
Es la ventana del editor principal y está compuesta de varias subventanas, llamadas vistas (views). Hay varios tipos de vistas en Unity y cada uno de ellos tiene una función específica. La organización de estas vistas en la ventana principal tiene varias configuraciones que puedes modificar con la opción Layout que se encuentra en la parte derecha de Toolbar. Al seleccionar esta opción puedes entre las siguientes opciones:



- 2 by 3: Es la configuración que se observa en la siguiente imagen, que consta de 2 (Scene y Game) por 3 (Hierarchy, Project e Inspector).

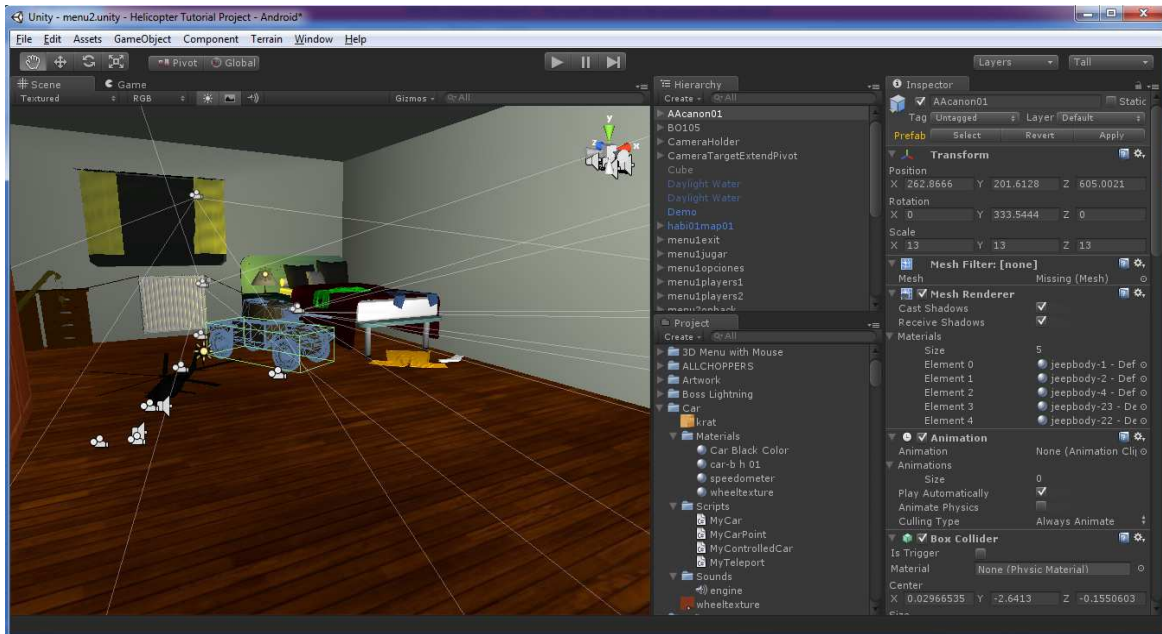


- 4 Split: Con esta configuración se muestra 4 vistas de Scene pero con diferentes ángulos y aparte Hierarchy, Project e Inspector. Esta configuración es más apropiada para el modelado en 3D debido a que te da una visión de alzado perfil y planta de los objetos 3D.

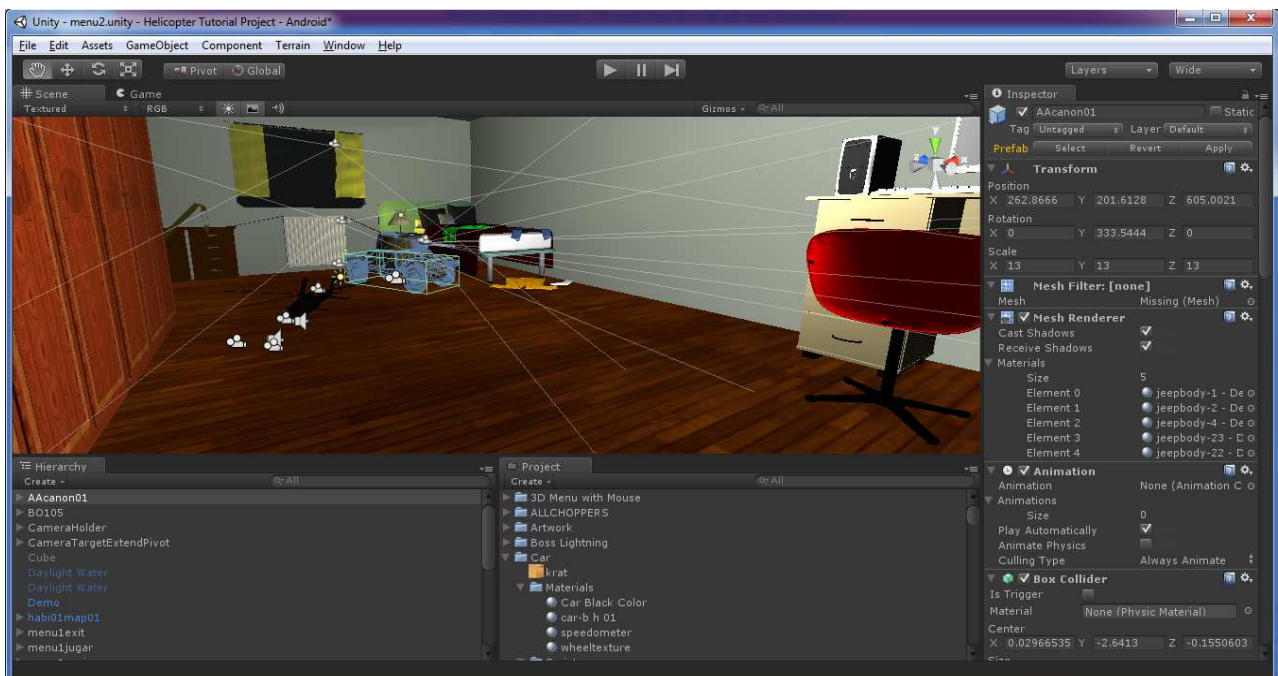




- Tall: Esta configuración por una vista más grande en la que puedes alternar entre Scene y Game



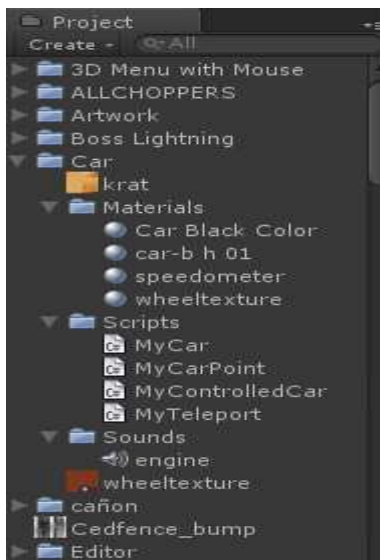
- Wide: Es muy similar a la anterior configuración, con la diferencia de la colocación de Hierarchy, Project e Inspector





- **Project View:**

En esta vista puedes encontrar todas las propiedades necesarias para crear tu juego, como scenes, scripts, 3D models, textures, audio files y prefabs. Todo proyecto en Unity tiene una carpeta donde se almacenan todas estas propiedades, también llamadas Assets, pero es muy recomendable no mover Project Assets usando el explorador del sistema operativo ya que se pueden romper vínculos entre datos del proyecto de Unity. Se debe usar siempre el Project View para organizar assets.

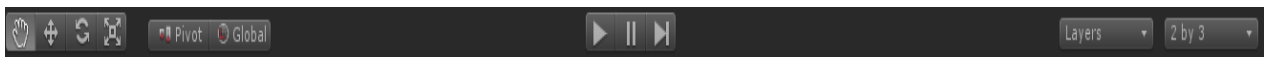


- **Hierarchy:**

Hierarchy contiene en todo momento los GameObjects que se encuentran en escena. Los objetos aparecen y desaparecen de la vista al mismo tiempo que los objetos se eliminan de la escena o se crean nuevos. Algunos de estos objetos son aplicaciones directas de los assets files como 3D models, y otros son aplicaciones de los Prefabs



- **Toolbar:**



La barra de herramientas básica consiste en cinco controles principales:

Transform tools:



Es usada en Scene View para seleccionar, mover, rotar y escalar

los objetos del juego.

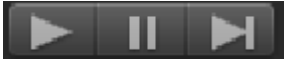
Transform Gizmo Toggles:



Afecta la forma del Scene View.



Control de reproducción:



Sirve para controlar la reproducción del juego en la vista Game. Puedes reproducir, parar y avanzar en el estado de la aplicación.

Layers Drop-down:



Controla los objetos que son mostrados en la vista Scene.

Layout Drop-down:



Controla la disposición de las vistas en la ventana principal como ya

hemos nombrado anteriormente.

- **Scene View:**

La vista Scene es una ventana interactiva en la cual puede seleccionar y posicionar los ambientes, el jugador, la cámara, y todos los demás GameObjects que intervengan en la aplicación. Para realizar todas estas funciones sobre Scene View debe utilizar Transform tools y el ratón del ordenador.





Utilizando la herramienta "mano" podrá mover la cámara de Scene View de izquierda a derecha, de arriba abajo pulsando el botón izquierdo del ratón. En caso de que no tener seleccionado la opción mano, obtendrá la misma función presionando la rueda del ratón.

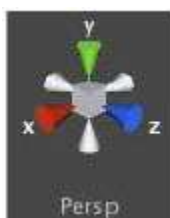


Manteniendo Alt y haciendo click puede girar sobre un punto central para obtener otro punto de vista.

Si pulsas sobre la vista Scene con el botón derecho del ratón y arrastras, la cámara Scene gira pero sobre el punto donde este situada la cámara Scene.

Para hacer Zoom en la vista Scene tiene dos posibilidades, o bien con la rueda del ratón, o pulsando Alt y botón derecho del ratón y arrastrando.

Otra función interesante, es la utilización de la tecla mayúscula (shift) al mismo tiempo que las anteriores combinaciones para que el efecto de Zoom o desplazamiento de la cámara sea mucho más rápido.



También es posible cambiar la cámara de la vista Scene a modo isométrico seleccionando el punto central o perspectiva, u obtener otro ángulo pulsando cualquiera de los ejes (X, Y, Z).

Otra manera de seleccionar las opciones de Transform tools es a través de los atajos de teclado. Las teclas Q, W, E, R corresponden a las herramientas hand, translate, rotate y scale respectivamente.



En el proceso de construcción de los juegos, se tendrá que ubicar varios objetos. Para ello, debemos usar Transform Tools en el toolbar para trasladar, rotar y escalar a GameObjects. Cada una de estas herramientas posee un Gizmo que aparece alrededor del objeto que esta seleccionado en el Scene View. Se puede usar el ratón y manipular cualquier Gizmo axis para modificar el componente Transform del GameObject, o también se pueden introducir valores directamente al componente Transform que se encuentra en el Inspector.




La vista Scene contiene una barra de control que nos permite ver la escena en varios modos de vista como textured, wireframe, RGB, overdraw, y otros. Esta barra te permitirá ver iluminaciones, oír los objetos con sonido, y elementos del juego.

- **Game View:**

La vista Game es generada desde la Main Camera, es decir, la cámara o las cámaras del juego. Es una vista representativa en la cual se puede observar el producto final sin tener que construir el proyecto y ejecutarlo.





La manera de controlar la reproducción del juego sobre esta vista es mediante la barra de control de reproducción de Toolbar.  Estos botones se utilizan para controlar el editor Play Mode, para así observar cómo funciona el producto final. Mientras se está ejecutando puedes observar en las demás vistas cómo evoluciona la aplicación y puedes realizar modificaciones sobre los objetos del juego ya que una vez parada la reproducción estos valores serán inicializados de nuevo con el valor antiguo.

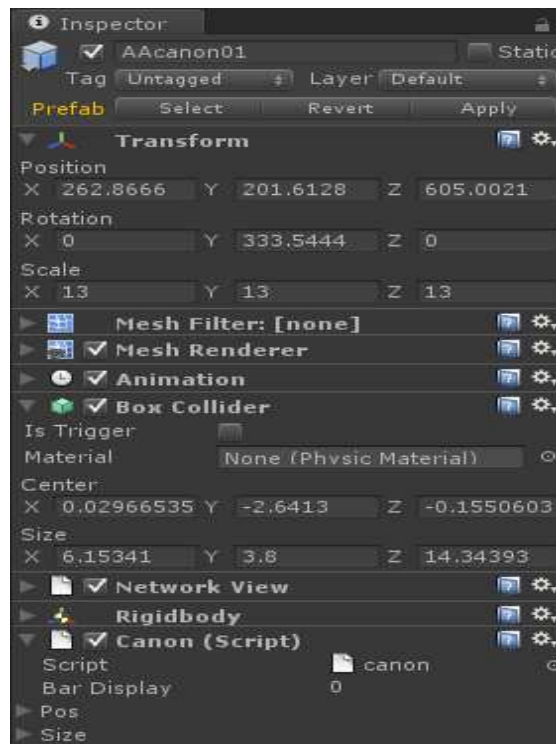


Mediante la barra de control de la vista Game puedes controlar las dimensiones del Game View con Free Aspect, maximizar al 100% la vista en modo Play, observar los Gizmos durante se está reproduciendo y mostrar las estadísticas de renderizado, que es muy útil para modificar y optimizar los gráficos.



- **Inspector:**

Los juegos y otras aplicaciones creadas por Unity están compuestos por varios GameObjects que contienen meshes, scripts, sounds, colliders y otros componentes. En la vista Inspector se muestra todos estos componentes que contienen el objeto seleccionado y todas sus propiedades.



De este modo cualquier propiedad de los componentes de los objetos pueden ser modificadas directamente sobre el Inspector sin cambiar el script. Otra utilidad del inspector es cambiar valores de las variables sobre Inspector durante el Modo Play para realizar pruebas.

La opción más importante en mi opinión es la de poder referenciar objetos a variables solo arrastrando. Es decir, puedes definir una variable en un Script como publica del tipo Transform o GameObject por ejemplo, y luego arrastrar y soltar sobre la variable en la vista inspector para así asignarlo como variable del Script.



Formación sobre el lenguaje JavaScript:

Vamos a explicar las características del lenguaje, realizando primero un recorrido por su historia y ver su creación y las diferencias y similitudes con java ya que puede llegar a confundirse y pensar que javascript es igual que java solo porque empiece por java.

- **Historia de JavaScript:**

JavaScript fue desarrollado originalmente por Brendan Eich de Netscape con el nombre de Mocha, el cual fue renombrado posteriormente a LiveScript, para finalmente quedar como JavaScript. El cambio de nombre coincidió aproximadamente con el momento en que Netscape agregó soporte para la tecnología Java en su navegador web Netscape Navigator en la versión 2.002 en diciembre de 1995. La denominación produjo confusión, dando la impresión de que el lenguaje es una prolongación de Java, y se ha caracterizado por muchos como una estrategia de mercadotecnia de Netscape para obtener prestigio e innovar en lo que eran los nuevos lenguajes de programación web.

«JavaScript» es una marca registrada de Oracle Corporation. Es usada con licencia por los productos creados por Netscape Communications y entidades actuales como la Fundación Mozilla.

Microsoft dio como nombre a su dialecto de JavaScript «JScript», para evitar problemas relacionados con la marca. JScript fue adoptado en la versión 3.0 de Internet Explorer, liberado en agosto de 1996, e incluyó compatibilidad con el Efecto 2000 con las funciones de fecha, una diferencia de los que se basaban en ese momento. Los dialectos pueden parecer tan similares que los términos «JavaScript» y «JScript» a menudo se utilizan indistintamente, pero la especificación de JScript es incompatible con la de ECMA en muchos aspectos.

Para evitar estas incompatibilidades, el World Wide Web Consortium diseñó el estándar Document Object Model (DOM, o Modelo de Objetos del Documento en español), que incorporan Konqueror, las versiones 6 de Internet Explorer y Netscape Navigator, Opera la versión 7, Mozilla Application Suite y Mozilla Firefox desde su primera versión.

En 1997 los autores propusieron JavaScript para que fuera adoptado como estándar de la European Computer Manufacturers 'Association ECMA, que a pesar de su nombre no es europeo sino internacional, con sede en Ginebra. En junio de 1997 fue adoptado como un estándar ECMA, con el nombre de ECMAScript. Poco después también como un estándar ISO.



Netscape introdujo una implementación de script del lado del servidor con Netscape Enterprise Server, lanzada en diciembre de 1994 (poco después del lanzamiento de JavaScript para navegadores web). A partir de mediados de la década de los 2000, ha habido una proliferación de implementaciones de JavaScript para el lado servidor. Node.js es uno de los notables ejemplos de JavaScript en el lado del servidor, siendo usado en proyectos importantes.

JavaScript se ha convertido en uno de los lenguajes de programación más populares en internet. Al principio, sin embargo, muchos desarrolladores renegaban el lenguaje porque el público al que va dirigido lo formaban publicadores de artículos y demás aficionados, entre otras razones. La llegada de Ajax devolvió JavaScript a la fama y atrajo la atención de muchos otros programadores. Como resultado de esto hubo una proliferación de un conjunto de frameworks y librerías de ámbito general, mejorando las prácticas de programación con JavaScript, y aumentando el uso de JavaScript fuera de los navegadores web, como se ha visto con la proliferación de entornos JavaScript del lado del servidor. En enero de 2009, el proyecto CommonJS fue inaugurado con el objetivo de especificar una librería para uso de tareas comunes principalmente para el desarrollo fuera del navegador web.

- [Características de JavaScript:](#)

Imperativo y estructurado

JavaScript soporta gran parte de la estructura de programación de C (por ejemplo, sentencias if, bucles for, sentencias switch, etc.). Con una salvedad, en parte: en C, el ámbito de las variables alcanza al bloque en el cual fueron definidas; sin embargo en JavaScript esto no es soportado, puesto que el ámbito de las variables es el de la función en la cual fueron declaradas. Esto cambia con la versión de JavaScript 1.7, ya que soporta block scoping por medio de la palabra clave let. Como en C, JavaScript hace distinción entre expresiones y sentencias. Una diferencia sintáctica con respecto a C es la inserción automática de punto y coma, es decir, en JavaScript los puntos y coma que finalizan una sentencia pueden ser omitidos.



Tipado dinámico

Como en la mayoría de lenguajes de scripting, el tipo está asociado al valor, no a la variable. Por ejemplo, una variable `x` en un momento dado puede estar ligada a un número y más adelante, religada a una cadena. JavaScript soporta varias formas de comprobar el tipo de un objeto, incluyendo duck typing. Una forma de saberlo es por medio de la palabra clave `typeof`.

Objetual

JavaScript esta formado casi en su totalidad por objetos. Los objetos en JavaScript son arrays asociativos, mejorados con la inclusión de prototipos (ver más adelante). Los nombres de las propiedades de los objetos son claves de tipo cadena: `obj.x = 10` y `obj['x'] = 10` son equivalentes, siendo la notación con punto azúcar sintáctico. Las propiedades y sus valores pueden ser creados, cambiados o eliminados en tiempo de ejecución. La mayoría de propiedades de un objeto (y aquellas que son incluidas por la cadena de la herencia prototípica) pueden ser enumeradas a por medio de la instrucción de bucle `for...in`. JavaScript tiene un pequeño número de objetos predefinidos como son `Function` y `Date`.

Evaluación en tiempo de ejecución

JavaScript incluye la función `eval` que permite evaluar expresiones como expresadas como cadenas en tiempo de ejecución. Por ello se recomienda que `eval` sea utilizado con precaución y que se opte por utilizar la función `JSON.parse()` en la medida de lo posible, pues resulta mucho más segura.

Prototipos

JavaScript usa prototipos en vez de clases para el uso de herencia. Es posible llegar a emular muchas de las características que proporcionan las clases en lenguajes orientados a objetos tradicionales por medio de prototipos en JavaScript.

Funciones como constructores de objetos

Las funciones también se comportan como constructores. Prefijar una llamada a la función con la palabra clave `new` crear una nueva instancia de un prototipo, que heredan propiedades y métodos del constructor (incluidas las propiedades del prototipo de `Object`). ECMAScript 5 ofrece el método `Object.create`, permitiendo la creación explícita de una instancia sin tener que heredar automáticamente del prototipo de `Object` (en entornos antiguos puede aparecer el prototipo del objeto creado como `null`).



La propiedad prototype del constructor determina el objeto usado para el prototipo interno de los

Nuevos objetos creados. Se pueden añadir nuevos métodos modificando el prototipo del objeto usado como constructor. Constructores predefinidos en JavaScript, como Array u Object, también tienen prototipos que pueden ser modificados. Aunque esto sea posible se considera una mala práctica modificar el prototipo de Object ya que la mayoría de los objetos en Javascript heredan los métodos y propiedades del objeto prototype, objetos los cuales pueden esperar que estos no hayan sido modificados.

- **Diferencias con Java:**

Compilador: Para programar en Java necesitamos un Kit de desarrollo y un compilador. Sin embargo, Javascript no es un lenguaje que necesite que sus programas se compilen, sino que éstos se interpretan por parte del navegador cuando éste lee la página.

Orientado a objetos: Java es un lenguaje de programación orientado a objetos. (Más tarde veremos que quiere decir orientado a objetos, para el que no lo sepa todavía) Javascript no es orientado a objetos, esto quiere decir que podremos programar sin necesidad de crear clases, tal como se realiza en los lenguajes de programación estructurada como C o Pascal.

Propósito: Java es mucho más potente que Javascript, esto es debido a que Java es un lenguaje de propósito general, con el que se pueden hacer aplicaciones de lo más variado, sin embargo, con Javascript sólo podemos escribir programas para que se ejecuten en páginas web.

Estructuras fuertes: Java es un lenguaje de programación fuertemente tipado, esto quiere decir que al declarar una variable tendremos que indicar su tipo y no podrá cambiar de un tipo a otro automáticamente. Por su parte Javascript no tiene esta característica, y podemos meter en una variable la información que deseemos, independientemente del tipo de ésta. Además, podremos cambiar el tipo de información de una variable cuando queramos.

Otras características: Como vemos Java es mucho más complejo, aunque también más potente, robusto y seguro. Tiene más funcionalidades que Javascript y las diferencias que los separan son lo suficientemente importantes como para distinguirlos fácilmente.



Formación sobre el lenguaje C Sharp:

Vamos a explicar las características del lenguaje, realizando primero un recorrido por su historia y ver su creación.

- **Historia de C, C++ y C#:**

El lenguaje de programación C# fue creado con el mismo espíritu que los lenguajes C y C++. Esto explica sus poderosas prestaciones y su fácil curva de aprendizaje. No se puede decir lo mismo de C y C++, pero como C# fue creado desde cero. Microsoft se tomó la libertad de eliminar algunas de las prestaciones más pesadas (como los punteros). Esta sección echa un vistazo a los lenguajes C y C++, siguiendo su evolución hasta C#.

El lenguaje de programación C fue diseñado en un principio para ser usado en el sistema operativo UNIX. C se usó para crear muchas aplicaciones UNIX. incluyendo un compilador de C. y a la larga se usó para describir el mismo UNIX. Su amplia aceptación en el mundo académico se amplió al mundo comercial y los proveedores de software como Microsoft y Borland publicaron compiladores C para los ordenadores personales. El API original para Windows fue diseñado para trabajar con código Windows escrito en C y el último conjunto de API básicos del sistema operativo Windows sigue siendo compatible con C hoy en día.

Desde el punto de vista del diseño. C carecía de un detalle que ya ofrecían otros lenguajes como Smalltalk: el concepto de objeto. Piense en un objeto como en una colección de datos y un conjunto de operaciones que pueden ser realizadas sobre esos datos. La codificación con objetos se puede lograr usando C pero la noción de objeto no era obligatoria para el lenguaje. Si quería estructurar su código para que simulara un objeto, perfecto. Si no, perfecto también. En realidad a C no le importaba. Los objetos no eran una parte fundamental del lenguaje, por lo que mucha gente no presto mucha atención a este estándar de programación.

Una vez que el concepto de orientación a objetos empezó a ganar aceptación, se hizo evidente que C necesitaba ser depurado para adoptar este nuevo modo de considerar al código. C++ fue creado para encarnar esta depuración. Fue diseñado para ser compatible con el anterior C (de manera que todos los programas escritos en C pudieran ser también programas C++ y pudieran ser compilados con el compilador de C++). La principal aportación a C++ fue la compatibilidad para el nuevo concepto de objeto. C++ incorporo compatibilidad para clases (que son "plantillas" de objetos) y permitió que toda una generación de programadores de C pensaran en términos de objetos y su comportamiento.



El lenguaje C++ es una mejora de C, pero aun así presenta algunas desventajas. C y C++ pueden ser difíciles de manejar. A diferencia de lenguajes fáciles de usar como Visual Basic, C y C++ son lenguajes de muy "bajo nivel" y exigen que mucho código para funcionar correctamente. Tiene que escribir su propio código para manejar aspectos como la gestión de memoria y el control de errores. C y C++ pueden dar como resultado aplicaciones muy potentes, pero debe asegurarse que el código funciona bien. Un error en la escritura del programa puede hacer que toda la aplicación falle o se comporte de forma inesperada. Como el objetivo al diseñar C++ era retener la compatibilidad con el anterior C, C++ fue incapaz de escapar de la naturaleza de bajo nivel de C.

Microsoft diseñó C# de modo que retuviera casi toda la sintaxis de C y C++. Los programadores que estén familiarizados con esos lenguajes pueden escoger el código C# y empezar a programar de forma relativamente rápida. Sin embargo, la gran ventaja de C# consiste en que sus diseñadores decidieron no hacerlo compatible con los anteriores C y C++. Aunque esto puede parecer un mal asunto, en realidad es una buena noticia. C# elimina las cosas que hacían que fuese difícil trabajar con C y C++. Como todo el código C es también código C++, C++ tenía que mantener todas las rarezas y deficiencias de C. C# parte de cero y sin ningún requisito de compatibilidad, así que puede mantener los puntos fuertes de sus predecesores y descartar las debilidades que complicaban las cosas a los programadores de C y C++.

- **Características de C#**

Sencillez de uso

C# elimina muchos elementos añadidos por otros lenguajes y que facilitan su uso y comprensión, como por ejemplo ficheros de cabecera, o ficheros fuentes IDL1 .12. Es por ello que se dice que C# es autocontenido. Además, no se incorporan al lenguaje elementos poco útiles, como por ejemplo macros, herencia múltiple u operadores diferentes al operador de acceso a métodos (operador punto) para acceder a miembros de espacios de nombres.

Modernidad

Al ser C# un lenguaje de última generación, incorpora elementos que se ha demostrado a lo largo del tiempo que son muy útiles para el programador, como tipos decimales o Booleanos, un tipo básico string, así como una instrucción que permita recorrer colecciones con facilidad(instrucción foreach). Estos elementos hay que simularlos en otros lenguajes como C++ o Java



Orientado a objetos

C# como lenguaje de última generación, y de propósito general, es orientado a objetos. C# no permite la inclusión de funciones ni variables globales que no estén incluidos en una definición de tipos, por lo que la orientación a objetos es más pura y clara que en otros lenguajes como C++. Además, C# soporta todas las características del paradigma de la programación orientada a objetos, como son la encapsulación, la herencia y el polimorfismo.

Orientado a componentes

La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular. La sintaxis de C# incluye por ejemplo formas de definir propiedades, eventos o atributos.

Recolección de basura

Como ya se comentó, todo lenguaje incluido en la plataforma .NET tiene a su disposición el recolector de basura del CLR. Esto implica que no es necesario incluir instrucciones de destrucción de objetos en el lenguaje.

Seguridad de tipos

C# incluye mecanismos de control de acceso a tipos de datos, lo que garantiza que no se produzcan errores difíciles de detectar como un acceso a memoria de ningún objeto, por ejemplo. Para ello, el lenguaje provee de una serie de normas de sintaxis, como por ejemplo no realizar conversiones entre tipos que no sean compatibles. Además, no se pueden usar variables no inicializadas previamente, y en el acceso a tablas se hace una comprobación de rangos para que no se excedan ninguno de los índices de la misma. Se puede controlar así mismo los desbordamientos en operaciones aritméticas, produciéndose excepciones cuando se produzcan.

Instrucciones seguras

Para evitar errores comunes como se producían programando en otros lenguajes, en C# se han impuesto una serie de restricciones en el uso de instrucciones de control más comunes. Por ejemplo, la evaluación de toda condición ha de ser una expresión condicional y no aritmética, como ocurría por ejemplo en C o en C++. Así se evitan errores por confusión del operador igualdad con el de asignación. Otra restricción que se impone en la instrucción de selección switch, imponiendo que toda selectora de la instrucción finalice con una instrucción break o goto que indique cuál es la siguiente acción a realizar.



Unificación de tipos

En C# todos los tipos derivan de una superclase común llamada System.Object, por lo que automáticamente heredarán todos los miembros definidos en esta clase. Es decir, son Objetos. A diferencia de Java, en C# esta característica también se aplica para los tipos básicos.

Extensión de los operadores básicos

Para facilitar la legibilidad de código y conseguir que los nuevos tipos de datos que se definan a través de las estructuras estén al mismo nivel que los elementos predefinidos en el lenguaje, al igual que C++ pero a diferencia de Java, C# permite redefinir el significado de la mayoría de los operadores (incluidos el de la conversión) cuando se apliquen a diferentes tipos de objetos.

Extensión de modificadores

C# ofrece, a través de los Atributos, la posibilidad de añadir a los metadatos del módulo resultante de la compilación de cualquier fuente información adicional a la generada por el compilador que luego podrá ser consultada en tiempo de ejecución a través de la biblioteca de reflexión de .NET.

Eficiente

En C#, todo el código incluye numerosas restricciones para garantizar su seguridad, no permitiendo el uso de punteros. Sin embargo, y a diferencia de Java, existen modificadores para saltarse esta restricción, pudiendo manipular objetos a través de punteros. Para ello basta identificar regiones de código con el identificador Unsafe, y podrán usarse en ellas punteros de forma similar a como se hace en C++. Esta característica puede resultar de utilidad en situaciones en las que se necesite gran velocidad de procesamiento.

Compatible

Para facilitar la migración de programadores de C++ o Java a C#, no sólo se mantiene una sintaxis muy similar a la de los dos anteriores lenguajes, sino que el CLR también ofrece la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objeto, tales como las DLLs de la API de Win32.



- Diferencias entre JavaScript y C# en Unity3d

Definición de funciones

En JavaScript las funciones son declaradas con la palabra clave `function`, mientras que en C# se usa el tipo de dato que la función nos va a devolver. Una función que no devuelve nada se declara con el tipo `Void`, pero otras que devuelven otros tipos se declaran con `String`, `float` o `bool` por ejemplo.

```
function Start () { /* stuff. */ } // JS  


---

void Start() { /* stuff. */ } // C#  
string MyFunc() { return "hello, world"; } // C#
```

Funciones genéricas

C# tiene funciones genéricas, que tienen asociado un parámetro con el cual identifican el tipo de función que son.

JavaScript

```
function Start()  
{  
  var someScript : ExampleScript = GetComponent(ExampleScript);  
  someScript.DoSomething();  
}
```

C#

```
void Start()  
{  
  var someScript = GetComponent<ExampleScript>();  
  // otra opción sería declararla así  
  var someScript = (ExampleScript) GetComponent(typeof(ExampleScript));  
  someScript.DoSomething();  
}
```



El comando For y Foreach

C# usa foreach y JavaScript for

```
for(var x in someList){ // JS
  x.someVar = true;
}
foreach(var x in someList) { // C#
  x.someVar = true;
}
```

La palabra clave new

C# necesita la palabra new al crear una estructura pero JavaScript no

```
var foo = Vector3(0, 0, 0); // JS
var foo = new Vector3(0, 0, 0); // C#
Instantiate(someObj, new Vector3(0, 0, 0), someRotation); // C#
```

Hacer Cast

```
var foo : GameObject = Instantiate(someObj...); // JS
var foo = Instantiate(someObj...) as GameObject; // JS
var foo = (GameObject) Instantiate(someObj...); // C#
var foo = Instantiate(someObj...) as GameObject; // C#
```



Declaración de variables

Se declaran de distinta manera como se puede apreciar

```
1. public int myPublicInt = 1; // a public var
2. int myPrivateInt = 2; // **private** access is default, if access is
   unspecified
3. public GameObject myObj; // a type is specified, but no value assigned
4.
5. // Javascript - type specification is not necessary:
6. var myPublicInt = 1; // **public** access is default, if unspecified
7. private var myPrivateInt = 2; // a private var
8. var myObj : GameObject;
```

Arrays multidimensionales

```
1. // C#:
2. int[,] = new int[16,16]; // 16x16 2d int array
3.
4. // Javascript:
5. var a = new int[16,16];
```

Definición de clases

Se hace de manera similar en los dos casos

```
1. // Javascript example
2. class MyClass extends MonoBehaviour {
3.
4.     var myVar = 1;
5.
6.     function Start() {
7.         Debug.Log("hello world!");
8.     }
9. }
10.
11. // C# example:
12. class MyClass : MonoBehaviour {
13.
14.     public int myVar = 1;
15.
16.     void Start() {
17.         Debug.Log("hello world!");
18.     }
19. }
```




2.2. Objetivos

El objetivo de este proyecto es la creación de una aplicación para el entretenimiento utilizando las ventajas ofrecidas por Unity 3d para este tipo de aplicaciones. Unity 3D es un programa para realizar juegos para internet, Play Station 3 y lo más importante para móviles Iphone, móviles con sistema Android y cualquiera que soporte Unity. En este proyecto la programación se realizará mediante los lenguajes JavaScript y C# además de los programas Autodesk 3ds Max y Maya para el diseño del escenario 3d.

Los objetivos a desarrollar, en una primera instancia, serían los siguientes:

- Estudio de las herramientas, lenguajes y programas a utilizar en el diseño de la aplicación
- Diseño de la aplicación.
- Implementación de la aplicación.
- Pruebas de la aplicación.
- Documentación de la aplicación para el correcto manejo del interface.

2.3. Fases del proyecto:

Etapas de desarrollo

- En primer lugar, realizaremos una etapa de formación para estudiar los lenguajes JavaScript y

C# para su posterior empleo en la aplicación. Realizaremos varios tutoriales para el estudio del programa Unity 3d donde realizaremos la implementación de la aplicación. También necesitaremos conocimientos básicos de 3ds max y maya para el

diseño 3d del escenario y objetos.

- En segundo lugar, es necesario realizar cuatro iteraciones (prototipos) para ir mejorando de manera progresiva nuestra aplicación.

Cada iteración constará de las siguientes fases:

- Análisis: Analizaremos los requisitos y requerimientos.
- Diseño del sistema: Se empezará por lo más general y se irá bajando hasta llegar al nivel más bajo de diseño. Seguiremos una metodología de Diseño Centrado en el Usuario.
- Implementación: A través de la herramienta Unity 3D en lenguaje C# y JavaScript.
- Pruebas: Periodo en el que se realizarán pruebas a nivel general del sistema, corrigiendo los errores y fallos que se vayan presentando.



- Documentación: Se harán informes en donde se muestre el progreso al terminar cada iteración del proyecto así como la elaboración de la documentación necesaria para el uso de la aplicación creada. Además de la entrega final (memoria del proyecto) con todos los datos.

Por último, realizaremos los manuales necesarios para la perfecta utilización de la aplicación.

2.4. Planificación:

| PLANIFICACIÓN | | | |
|-------------------------|----------------|------------|----------|
| Nombre Etapa | Fecha comienzo | Fecha fin | Duración |
| Formación | 21/11/2012 | 21/11/2010 | 1 mes |
| Iteración 1 | 21/12/2012 | 21/01/2011 | 2 meses |
| Iteración 2 | 21/02/2013 | 21/02/2011 | 1 mes |
| Iteración 3 | 21/03/2013 | 21/03/2011 | 1 mes |
| Iteración 4 | 21/04/2013 | 21/04/2011 | 1 mes |
| Documentación y memoria | 21/5/2013 | 21/05/2011 | 1 mes |



3 Desarrollo:

3.1. Formación en Unity 3D:

En los apartados anteriores ya hemos explicado el funcionamiento básico del programa Unity 3D, y a continuación, vamos a explicar las clases más comunes que se utilizan para construir una aplicación en Unity.

Las clases son las siguientes:

1. Collision
2. ContactPoint
3. Debug
4. GUI
5. Input
6. Object
 - a. Component
 - i. Behaviour
 1. AudioSource
 2. AudioListener
 3. GUIElement
 - a. GUIText
 - b. GUITexture
 4. Light
 5. MonoBehaviour
 - ii. Collider
 - iii. ParticleAnimator
 - iv. ParticleEmitter
 - v. Rigidbody
 - vi. Transform
 - b. GameObject
 - c. Texture
7. Ray
8. RaycastHit
9. Screen
10. Time
11. Touch
12. Vector3



Después de explicar las clases, comentaremos el proceso de compilación y ejecución de la aplicación en un dispositivo con sistema operativo Android, es decir, el proceso de pasar la aplicación de Unity al propio móvil con Android.

Clases de Unity:

1) Collision:

Describe las colisiones de los objetos. La información de las colisiones es pasada por los eventos:

- Collider.OnCollisionEnter:

Este evento es llamado cuando el collider/rigidbody ha comenzado a tocar a otro rigidbody/collider. A diferencia de `OnTriggerEnter`, en `OnCollisionEnter` es pasado como parámetro el `Collision` class y no un `Collider`. La `Collision` class contiene información sobre puntos de contacto (contact points), velocidad de impacto (impact velocity), etc.

- Collider.OnCollisionStay:

Es llamado una vez por frame para todo collider/rigidbody que está tocando otro rigidbody/collider. A diferencia de `OnTriggerStay`, `OnCollisionStay` pasa la clase `Collision` y no un `Collider`.

- Collider.OnColliderExit:

Este evento, es llamado cuando el collider/rigidbody ha dejado de tocar otro rigidbody/collider. Como en los anteriores, este evento también pasa la clase `Collision` en vez de un `Collider`.



La clase Collision tiene unas cuantas variables que contienen información sobre la colisión :

| | |
|------------------|---|
| relativeVelocity | La velocidad lineal relativa de los dos objetos de la colisión (solo lectura) |
| rigidbody | El Rigidbody que golpeamos, esta variable será nula si el collider golpeado no contiene el componente rigidbody |
| collider | El Collider golpeado |
| transform | El componente Transform que es golpeado |
| gameObject | /gameObject/ es el objeto con el que estamos chocando |
| contacts | Los puntos de contacto generados por la física de los objetos. |

Veamos ahora el diagrama de la clase Collision:





2) ContactPoint:

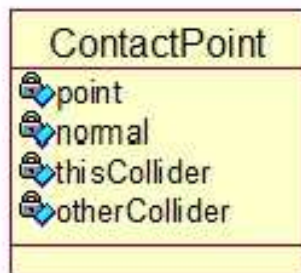
Describe el punto de contacto cuando sucede una colisión.

Los puntos de contacto son almacenados en la estructura Collision.

Las variables de información son:

| | |
|---------------|-----------------------------------|
| point | EL punto de contacto |
| normal | La normal de el punto de contacto |
| thisCollider | El primer collider en contacto |
| otherCollider | El otro collider en contacto |

Veamos el diagrama de la clase:



3) Debug:

Es la clase que contiene los métodos para realizar un debug de la manera más sencilla desarrollando un juego.

Las funciones de la clase son:

| | |
|------------|---|
| DrawLine | Dibuja una línea en el punto en el cual comienza y acaba con color |
| DrawRay | Dibuja una línea de start to start + dir con color |
| Break | Pausa el editor |
| Log | Mandas un mensaje a la consola de Unity |
| LogError | Es una variante de Debug.Log que manda un mensaje de error a la consola. |
| LogWarning | Es una variante también, que manda un mensaje de aviso a la consola de Unity. |

Veamos el diagrama de la clase:



4) GUI:

La clase GUI es el interfaz para los GUI de Unity con posicionamiento manual.

Su constructor es GUI (static function GUI () : GUI)

Las variables de la clase son:

| | |
|-----------------|--|
| skin | El skin global para usar |
| color | Tinte de color global para la GUI |
| backgroundColor | Color global de fondo para todos los elementos de laGUI |
| contentColor | Color para todos los textos renderizados por GUI |
| changed | Devuelve true si algún control cambia de valor |
| enabled | Esta el GUI habilitado? |
| matrix | El GUI transforma matrix |
| tooltip | Una descripción de la posición del raton sobre los elementos GUI |

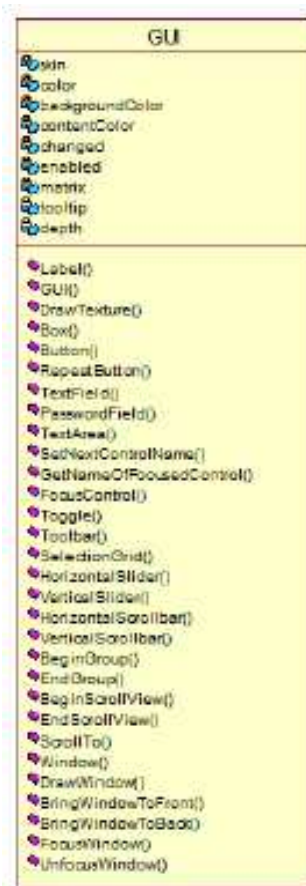


Las funciones de la clase:

| | |
|-------------------------|--|
| Label | Crea un texto o un texture label en la pantalla |
| DrawTexture | Dibuja una textura dentro de un rectángulo |
| Box | Crea un graphical box |
| Button | Crea un simple press button. |
| RepeatButton | Crea un botón que se activa siempre que el usuario mantenga pulsado. |
| TextField | Crea un text field donde el usuario puede modificar el texto |
| PasswordField | Crea un text field donde el usuario puede introducir la contraseña |
| TextArea | Crea un multi-line text area, donde el usuario puede editar texto |
| SetNextControlName | Pone el nombre para el próximo control |
| GetNameOfFocusedControl | Adquiere el nombre del control llamado. |
| FocusControl | Mueve el foco de teclado para el control nombrado |
| Toggle | Crea un on/off toggle button |
| Toolbar | Crea una barra de herramientas |
| SelectionGrid | Crea una red de botones |
| HorizontalScrollbar | Crea una barra de desplazamiento horizontal |
| VerticalScrollbar | Crea una barra de desplazamiento vertical |
| BeginGroup | Comienza un grupo, y debe ir acompañada con una llamada a EndGroup |
| EndGroup | Finaliza un grupo |
| BeginScrollView | Comienza un scrolling view dentro de GUI |
| EndScrollView | Termina un scrolling view |
| ScrollTo | Desplaza todos los scrollviews adjuntando por lo que tratar de hacer lugar visible |
| Window | Crea un popup window |
| DragWindow | Hace una ventana movable |
| BringWindowToFront | Traiga una ventana específica al frente de las ventanas flotantes. |
| BringWindowToBack | Lleve una ventana específica al fondo de las ventanas |
| FocusWindow | Hacer que una ventana se convierta en la ventana activa |
| UnfocusWindow | Eliminar el foco sobre todas las ventanas |



Veamos el diagrama de clase de GUI donde podremos observar sus variables (atributos) y funciones:



5) Input:

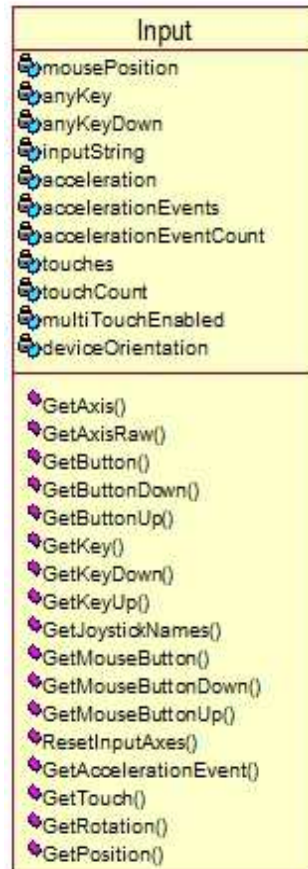
Interfaz en el input system (sistema de entrada)

Esta clase es usada para leer los ejes establecidos en el Input Manager, y para acceder a los datos del multi-touch/accelerometro de los dispositivos móviles

Para leer un eje se usa Input.GetAxis con uno de los siguientes ejes por defecto: "Horizontal" y "vertical" son asignados en el joystick, A, W, S, D y las flechas del teclado. "Mouse X" y "Mouse Y" son asignados en el ratón. "Fire1", "Fire2", "Fire3" son asignados para las teclas Ctrl, Alt, Cmd y tres botones del ratón o joystick. Nuevas ejes de entrada (input) pueden ser añadidos en el Input Manager.



A continuación veremos dos imágenes, una en la que se muestra el interfaz del InputManager y otra en la cual se verá el diagrama de la clase.



Como se puede observar en la anterior imagen, en el InputManager se encuentran todas las funciones asignadas, 17 en total, pero que pueden añadirse más si modificamos el valor de "size" para asignar una nueva función si lo considerásemos oportuno.

Si tu estas usando input para cualquier modo de comportamiento de un movimiento usa Input.GetAxis. Esta te proporcionará una entrada configurable que puede ser asignada al teclado, joystick o el ratón. Usa Input.GetButton para acciones como eventos y no lo uses para movimientos ya que Input.GetAxis hará el código del script más pequeño y simple.



Input en dispositivos móviles:

Los dispositivos iOS y Android están capacitados para captar múltiples toques de dedo en la pantalla simultáneamente. Puedes acceder a los datos del estado de ese toque de dedo (finger touching screen) en el anterior frame accediendo al array de la propiedad `Input.touches`.

Como un dispositivo se mueve, su hardware acelerómetro envía una señal lineal de aceleración que cambia en los tres ejes primarios de las tres dimensiones del espacio. Puedes utilizar esa información para detectar la orientación actual del dispositivo y si esta orientación cambia.

El hardware de aceleración envía los valores de orientación de los ejes del dispositivo en forma de fuerzas G, de tal forma que un valor de 1.0 representa una fuerza de +1G en ese eje, mientras que un valor de -1.0 representa una fuerza de -1G.

Para obtener los datos del acelerómetro puedes leer la propiedad `Input.acceleration`, aunque también puedes utilizar la propiedad `Input.deviceOrientation` para saber exactamente la orientación del dispositivo en el eje que quieras.

Las variables de la clase son:

| | |
|-------------------------------------|---|
| <code>mousePosition</code> | La posición del ratón en coordenadas pixel. |
| <code>anyKey</code> | Esta alguna tecla o botón del ratón siendo pulsada? |
| <code>anyKeyDown</code> | Devuelve true en el primer frame que el usuario presione cualquier botón. |
| <code>inputString</code> | Devuelve la input de teclado entrada en este frame |
| <code>acceleration</code> | Última aceleración del dispositivo en los tres ejes |
| <code>accelerationEvents</code> | La lista de compases de aceleración en el anterior frame |
| <code>accelerationEventCount</code> | Numero de aceleraciones en el anterior frame |
| <code>touches</code> | Devuelve la lista de objetos que representan el estado de todos los touches en el último frame. |
| <code>touchCount</code> | El número de touches. |
| <code>multiTouchEnabled</code> | Indica si el sistema se encarga de multiTouch |
| <code>deviceOrientation</code> | Orientación del dispositivo física |



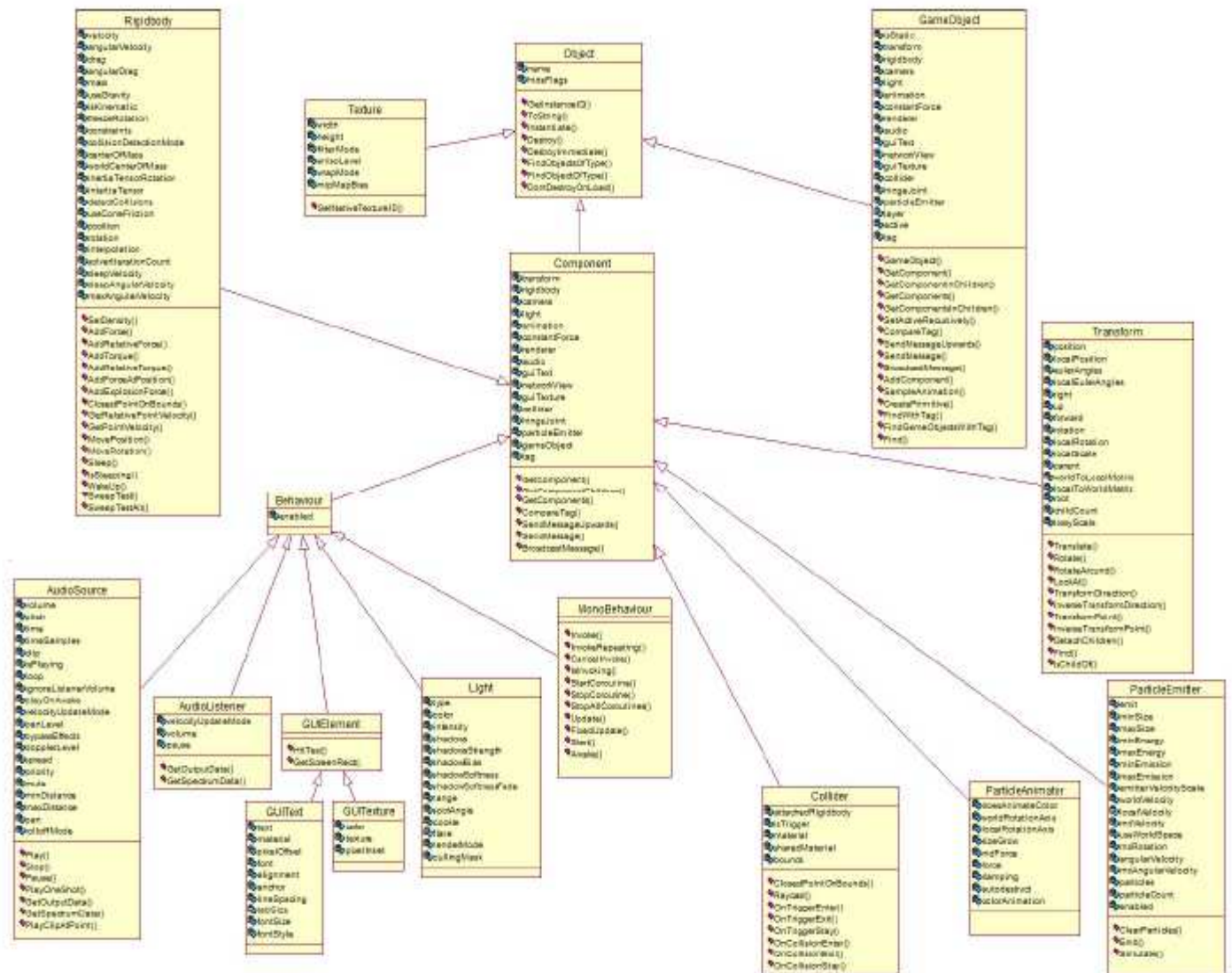
Las funciones de la clase son:

| | |
|----------------------|--|
| GetAxis | Devuelve el valor del eje virtual identificado por axisName |
| GetAxisRaw | Devuelve el valor del eje virtual identificado por axisName, sin aplicar filtro smoothing. |
| GetButton | Devuelve true mientras el botón virtual este pulsado |
| GetButtonDown | Devuelve true durante el frame que el botón se encuentra pulsado |
| GetButtonUp | Devuelve true en el primer frame que el usuario deja de pulsar el botón. |
| GetKey | Devuelve true mientras el usuario presiona la tecla identificada por el nombre |
| GetKeyDown | Devuelve true durante el frame que el usuario comienza a presionar la tecla |
| GetKeyUp | Devuelve true durante el frame que el usuario deja de presionar la tecla |
| GetJoystickNames | Devuelve un array con la descripción de los joystick conectados |
| GetMouseButton | Indica si el botón está siendo pulsado |
| GetMouseDown | Devuelve true durante el frame que el usuario presiona el botón. |
| GetMouseUp | Devuelve true durante el frame que el usuario deja de presionar el botón. |
| ResetInputAxes | Resetea todas las entradas (inputs) |
| GetAccelerationEvent | Devuelve la descripción de la aceleración del ultimo frame |
| GetTouch | Devuelve el objeto que representa el estado del touch |
| GetRotation | |
| GetPosition | |



6) Object:

Es la clase base de todos los objetos que Unity puede hacer referencia. Cualquier variable que derive de un objeto podrá verse en el inspector permitiéndote dar un valor en el interfaz, sin tener que modificar el script.





En esta imagen vemos el diagrama de clases de la clase Object, en el cual vemos las relaciones de herencia con las clases más importantes y utilizadas en nuestro proyecto, como son Rigidbody, Transform, GameObject, Texture, Component, y muchas más.



Las variables de la clase son:

| | |
|-----------|--|
| name | El nombre del objeto |
| hideFlags | Si el objeto se oculta, salva la escena o modifica por el usuario. |

Las funciones

| | |
|---------------|--|
| GetInstanceID | Devuelve el identificador de la instancia del objeto |
| ToString | Devuelve el nombre de el objeto |

Las funciones de la clase (class functions):

| | |
|-------------------|--|
| Operador bool | Existe el objeto? |
| Instantiate | Clona el objeto original y devuelve el clone |
| Instantiate.<T> | |
| Destroy | Elimina un gameobject, componente o asset |
| DestroyImmediate | Destruye el objeto inmediatamente. |
| FindObjectsOfType | Devuelve una lista de objetos cargados del mismo tipo |
| FindObjectOfType | Devuelve el primer objeto activo del mismo tipo |
| Operador == | Compara si dos objetos hacen referencia al mismo |
| Operador != | Compara si dos objetos no hacen referencia al mismo. |
| DontDestroyOnLoad | Hace que el objeto de destino no se destruya automáticamente cuando se carga una nueva escena. |

AnimationClip, AssetBundle AudioClip Component, Flare, Font, GameObject, Material, Mesh, PhysicMaterial, ScriptableObject, Shader, TerrainData, TextAsset y Texture, estas son las clases que heredan de la clase Object, por lo que solo explicaremos las clases que hemos utilizado en la creación de nuestra aplicación.



a) Component(hereda de Object):

Es la clase base para todos los componentes que van unidos a un gameObject. Esta clase hereda todos los miembros de la clase Object anteriormente nombrados.

Las variables son:

| | |
|-----------------|--|
| transform | El Transform que está unido al GameObject (null si no hay unido) |
| rigidbody | El Rigidbody que está unido al GameObject (null si no hay unido) |
| camera | El Camera que está unido al GameObject (null si no hay unido) |
| light | El Light que está unido al GameObject (null si no hay unido) |
| animation | El Animation que está unido al GameObject (null si no hay unido) |
| constantForce | El ConstantForce que está unido al GameObject (null si no hay unido) |
| renderer | El Renderer que está unido al GameObject (null si no hay unido) |
| audio | El AudioSource que está unido al GameObject (null si no hay unido) |
| guiText | El GUIText que está unido al GameObject (null si no hay unido) |
| networkView | El NetworkView que está unido al GameObject (null si no hay unido) |
| guiTexture | El GUITexture que está unido al GameObject (null si no hay unido) |
| collider | El Collider que está unido al GameObject (null si no hay unido) |
| hingeJoint | El HingeJoint que está unido al GameObject (null si no hay unido) |
| particleEmitter | El ParticleEmitter que está unido al GameObject (null si no hay unido) |
| gameObject | El game object al que están unidos los componentes |
| tag | El tag de el game object. |



Las funciones son:

| | |
|------------------------|---|
| GetComponent | Devuelve el componente del tipo o del nombre pasado por parámetro si el objeto tiene unido alguno, si no, devuelve null |
| GetComponentInChildren | Devuelve el componente del tipo seleccionado del gameObject o de alguno de sus hijos usando la búsqueda en profundidad |
| GetComponentInChildren | Devuelve los componentes del tipo seleccionado del GameObject o alguno de sus hijos |
| GetComponents | Devuelve todos los componentes del tipo seleccionado que tiene el GameObject |
| CompareTag | Este GameObject está etiquetado como tag? |
| SendMessageUpwards | Invoca al método llamado methodName en cualquier script de ese gameObject y en todos los ancestros de el gameObject |
| SendMessage | Invoca al método llamado methodName en cualquier script de ese gameObject |
| BroadcastMessage | Invoca al método llamado methodName en cualquier script de ese gameObject y todos sus hijos. |

En Unity hay varias clases que heredan de la clase base Component. Estas son Behaviour, Cloth, Collider, Joint, MeshFilter, OcclusionArea, ParticleAnimator, ParticleEmitter, Renderer, Rigidbody, TextMesh, Transform y Tree.

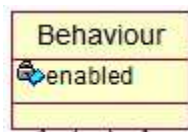


b) Behaviour (hereda de Component):

Behaviour son componentes que se pueden habilitar o deshabilitar. Solo contienen una variable, `enabled`, y con ella habilitan o deshabilitan los componentes.

Poseen muchos miembros heredados tanto de `Component` como de `Object`.

La siguiente imagen corresponde con el diagrama de la clase Behaviour.



(1) AudioSource (hereda de Behaviour):

A un `GameObject` se le une un componente `AudioSource` para reproducir música de fondo con tecnología 3D. Para reproducir sonidos 3D necesitas tener un `AudioListener`. El `AudioListener` normalmente está unido en la cámara que el usuario utiliza. Los sonidos estéreo son reproducidos siempre sin atenuación por la distancia.

Puede reproducir un único clip de audio usando `Play`, `Pause`, y `Stop`. También, puedes ajustar el volumen del sonido con la propiedad `volume`. Se pueden reproducir múltiples sonidos en uno utilizando `PlayOneShot`. Puedes reproducir sonidos en una posición estática en el espacio 3D usando `PlayClipAtPoint`.



Las variables de la clase son:

| | |
|----------------------|---|
| volume | El volumen del audio source |
| pitch | El pitch del audio source |
| time | Reproducción de la posición en segundos |
| timeSamples | Reproducción de la posición en PCM samples. |
| clip | El AudioClip por defecto para reproducir |
| isPlaying | El clip se está reproduciendo correctamente? |
| loop | El clip de audio esta en un bucle? |
| ignoreListenerVolume | hace que audio source no tenga en cuenta el volumen de el audio listener |
| playOnAwake | Si le asignamos true, el audio source automáticamente empezara a reproducirse dentro de awake |
| velocityUpdateMode | Si audio source debe ser actualizada en fijos o dinamicos |
| panLevel | Determina cuanta tecnología 3D va a tener efecto en el canal |
| bypassEffects | El efecto bypassEffect |
| dopplerLevel | Indica la escala doppler de el AudioSource |
| spread | Establece el ángulo de propagación del sonido. |
| priority | Establece la prioridad de el AudioSource |
| mute | Silencia o no el audio. Para silenciar mute=0 |
| minDistance | La mínima distancia a la que dejara de aumentar el volumen |
| maxDistance | La distancia máxima a la que el sonido dejara de atenuarse |
| pan | Establece canales pan de forma lineal. Solo funciona en 2D clip |
| rolloffMode | Establece o obtiene el modo en el que audiosource se atenúa fuera de distancia |

Las funciones de la clase son:

| | |
|-----------------|--|
| Play | Reproduce el clip |
| Stop | Para de reproducir el clip |
| Pause | Pausa la reproducción |
| PlayOneShot | Reproduce un AudioClip |
| GetOutputData | Devuelve un bloque de datos output de audio que se están reproduciendo |
| GetSpectrumData | Devuelve un bloque de datos spectrum de audio que se están reproduciendo |



(2) AudioListener (hereda de Behaviour):

Representa a un listener en el espacio 3D. Este componente graba todos los sonidos que se reproducen a su alrededor y los reproduce para que el usuario los escuche. Solo se puede tener un listener en cada escena.

Posee una variable `velocityUpdateMode` con la que el audio listener puede ser actualizado en fijo o dinámico. Con `volume` puedes controlar el volumen del juego y con `pause` puedes parar el estado del audio. Si `pause` es `true` el audio no se oirá.

El diagrama es el siguiente:



(3) GUIElement (hereda de Behaviour):

Es la clase básica para imágenes y texto mostrados en la interfaz de usuario (GUI). Esta clase contiene todas las funcionalidades de cualquier elemento GUI.

Las funciones son:

| | |
|----------------------------|---|
| <code>HitTest</code> | Es un punto en la pantalla dentro del elemento |
| <code>GetScreenRect</code> | Devuelve un rectángulo que delimita el <code>GUIElement</code> en coordenadas de pantalla |

Esta clase hereda dos clases importantes:



(a) `GUIText` (hereda de `GUIElement`):

Es una frase de texto que se muestra en un interfaz (GUI).

Las variables son:

| | |
|--------------------------|---|
| <code>text</code> | El texto para mostrar |
| <code>material</code> | El materia a usar en el renderizado |
| <code>pixelOffset</code> | Los pixel de desplazamiento del texto |
| <code>font</code> | La tipografía de letra para el texto |
| <code>alignment</code> | La alineación del texto |
| <code>anchor</code> | El anclaje del texto |
| <code>lineSpacing</code> | El espacio interlineal |
| <code>tabSize</code> | El tamaño del tabulador |
| <code>fontSize</code> | El tamaño de fuente |
| <code>fontStyle</code> | El estilo de fuente (cursiva, negrita, ...) |

(b) `GUITexture` (hereda de `GUIElement`):

Es una textura usada en 2D GUI mediante una imagen.

Contiene tres variables:

| | |
|-------------------------|---|
| <code>color</code> | El color de la GUI texture |
| <code>texture</code> | La textura a dibujar en pantalla |
| <code>pixelInset</code> | Es usada para ajustar por tamaño y posición |

4) `Light` (hereda de `Behaviour`):

Esta clase se usa para controlar el aspecto de las luces en Unity. Las propiedades se muestran también en el inspector, y lo más usual es cambiar y controlar los valores de la luz mediante el inspector, aunque otras veces puede ser de utilidad modificar estos valores en código.



Las variables son:

| | |
|--------------------|--|
| type | El tipo de luz |
| color | El color de la luz |
| intensity | La intensidad que es multiplicada por el color de la luz |
| shadows | Como quieres que muestre las sombras? |
| shadowStrength | La fuerza de las sombras de la luz |
| shadowBias | Sombra en Bias |
| shadowSoftness | La suavidad de las sombras de las luces direccionales |
| shadowSoftnessFade | La velocidad de suavidad de las sombras |
| range | El rango al que afecta la luz |
| spotAngle | Los ángulos de inclinación de la luz |
| cookie | la textura cookie proyectada por la luz |
| flare | El flare asset a usar para esta luz |
| renderMode | El modo de renderizar la luz |
| cullingMask | Es usado para iluminar partes de la escena selectivamente. |

5) MonoBehaviour (hereda de Behaviour):

Es la clase de la que derivan o heredan todos los script de la aplicación.

Cuando usas Javascript todos los scripts automáticamente derivan de MonoBehaviour. Sin embargo, cuando utilizamos C# o Boo, tenemos que especificar que hereda o deriva de MonoBehaviour.

La única variable que maneja esta clase es useGUILayout, que se usa para saltar la fase de diseño de la GUI, deshabilitando esta variable.



Las funciones de la clase son:

| | |
|-------------------|--|
| Invoke | Invoca el método pasado por parámetro en x segundos |
| InvokeRepeating | Invoca el método pasado por parámetro en x segundos |
| CancelInvoke | Cancela todas las llamadas a métodos en este MonoBehaviour |
| IsInvoking | Hay alguna invocación de methodName pendiente? |
| StartCoroutine | Empieza un coroutine |
| StopCoroutine | Para todos los coroutine llamados methodName que están corriendo en este behaviour |
| StopAllCoroutines | Para todos los coroutine que están corriendo en este behaviour |

De las funciones Overridable vamos a explicar las funciones más importantes y más utilizadas en este proyecto.

- Update:

Es la función llamada cada frame, siempre y cuando MonoBehaviour este habilitado, y su uso más común es para implementar el modo de comportamiento del juego.

- FixedUpdate:

Esta función es llamada cada fotograma fijo de imágenes por segundo, si el MonoBehaviour está habilitado. Puede resultar interesante su uso, en vez de Update, cuando se trabaja con un rigidbody, si quiere aplicar una fuerza cada fotograma fijo, por ejemplo, hágalo dentro de FixedUpdate, en vez de cada frame dentro de Update.

- Awake:

Esta función es llamada cuando la instancia del script se está cargando. Suele utilizarse para inicializar variables o el estado del juego antes de que el juego comience.



- Start:

Es llamada justo antes de que cualquier método Update sea llamado por primera vez. Es llamado solo una vez en la vida del Behaviour.

La mayoría del resto de funciones como OnMouseEnter, OnMouseOver, OnCollisionEnter, etc, ya han sido explicadas anteriormente por lo que no serán tratadas de nuevo. La diferencia entre Start y Awake es que Start es llamado solamente si la instancia del script está habilitada.

- ii) Collider (hereda de Component):

Es la clase fundamental de todos los colliders. Los colliders son objetos que detectan colisiones de otros objetos. Hay cuatro tipos de colliders:

- BoxCollider: Es un collider de forma cuadrangular que posee dos variables para regular su tamaño y posición que son center y size.
- SphereCollider: Es un collider con forma esférica que tienes dos variables, center y radius para configurar su tamaño y posición.
- CapsuleCollider: Es un collider con forma de capsula que tiene como variables de la clase: center, radius, height y direction, para configurar la forma.
- MeshCollider: MeshCollider es un collider que se adapta a la forma del objeto al cual se aplica.

La clase collider contiene las siguientes variables:

| | |
|-------------------|--|
| attachedRigidbody | El rigidbody al cual está unido el collider |
| isTrigger | Es trigger este collider? |
| material | El material usado por el collider |
| sharedMaterial | El material para compartir de el collider |
| bounds | El volumen que delimita el colisionador (collider) |



Las funciones de la clase:

| | |
|----------------------|---|
| ClosestPointOnBounds | El punto más cercano la caja de delimitación del collider adjunto |
| Raycast | Lanza un ray que pasa por alto todos los collider excepto este |

iii) ParticleAnimator (hereda de component):

Este componente es el encargado de mover las partículas en el tiempo, se usa para aplicar viento, fuerzas de arrastre y color a los sistemas de partículas. Esta clase es un script para un componente de animación de partículas.

Las variables de la clase son:

| | |
|-------------------|---|
| doesAnimateColor | Hace ciclar su color durante toda la vida de la partícula? |
| worldRotationAxis | Alrededor de que eje del mundo de las partículas rota |
| localRotationAxis | Alrededor de que eje local de las partículas rota. |
| sizeGrow | Como las partículas aumentan el tamaño durante su vida |
| rndForce | Una fuerza aleatoria es aplicada a las partículas en todo frame |
| force | La fuerza aplicada a las partículas en todo frame |
| damping | Como de lentas son cada vez más las partículas |
| autodestruct | Destruye el gameobject de este animador de partículas |
| colorAnimation | El color de las partículas que utilizaran durante su vida |

iv) ParticleEmitter (hereda de component):

Es un script de interfaz para emitir partículas. Tanto ParticleEmitter como ParticleAnimator son scripts que se pueden modificar en el interfaz de Unity en la ventana inspector o en el script.



Las variables de la clase son:

| | |
|----------------------|---|
| emit | Permite a las partículas ser emitidas automáticamente cada frame |
| minSize | El mínimo tamaño que puede tener cada partícula |
| maxSize | El máximo tamaño que puede tener cada partícula |
| minEnergy | El mínimo tiempo de vida de cada partícula |
| maxEnergy | El máximo tiempo de vida de cada partícula |
| minEmission | El mínimo número de partículas emitidas por segundo |
| maxEmission | El máximo número de partículas emitidas por segundo |
| emitterVelocityScale | Aumenta la velocidad de las partículas emitidas |
| worldVelocity | La velocidad inicial de las partículas en los ejes X, Y, Z globales |
| localVelocity | La velocidad inicial de las partículas en los ejes X, Y, Z locales |
| rndVelocity | Una velocidad aleatoria que es añadida sobre los ejes X, Y, Z |
| useWorldSpace | si está habilitada, cuando mueves el editor las partículas no se mueven, si está deshabilitada, si que se mueven. |
| rndRotation | Si está habilitada, las partículas se mueven con rotación aleatoria |
| angularVelocity | La velocidad angular de las nuevas partículas en grados |
| rndAngularVelocity | Una nueva velocidad angular que modifica las nuevas partículas |
| particles | Devuelve una copia de todas las partículas y asigna un array de todas las partículas que están actualmente corriendo. |
| particleCount | El número de partículas que corren. |
| enabled | Habilita o deshabilita la emisión de partículas |

Las funciones son:

| | |
|----------------|--|
| ClearParticles | Elimina todas las partículas del sistema |
| Emit | Emite un número de partículas |
| Simulate | Avance del sistema de partículas simulado con tiempo determinado |



v) Rigidbody (herada de component):

La clase Rigidbody controla la posición de los objetos en una simulación física. Los componentes Rigidbody tienen la función de controlar la posición de los objetos, de controlar si al objeto le afectará o no la fuerza gravitatoria, y calcular como responden los objetos ante colisiones.

Cuando manipulas los parámetros de un Rigidbody, es recomendable trabajar sobre la función FixedUpdate como ya comentamos anteriormente.

Un punto a tener en cuenta cuando utilizamos los rigidbodies es:

Si la simulación se observa en cámara lenta y no solida:

Es un problema de escala. Cuando el mundo del juego es tan grande como parece se mueve muy despacio. Asegúrese de que todos tus modelos se encuentran el tamaño del mundo real. Por ejemplo, si un coche mide aproximadamente 4 metros de largo y en el juego lo ponemos a 2 metros, el objeto mantendrá el peso del objeto de dos metros y la simulación de la caída pos corresponderá con la de un coche de dos metros y parecerá que cae a cámara lenta. Por eso es mejor mantener el tamaño de los objetos del mundo real.



Las variables de la clase son:

| | |
|------------------------|---|
| velocity | El vector de velocidad del rigidbody |
| angularVelocity | La velocidad angular del rigidbody |
| drag | El arrastre |
| angularDrag | El arrastra angular de el objeto |
| mass | El peso o masa del objeto |
| useGravity | Controlas si le afecta o no la gravedad |
| isKinematic | Controlas si la física afecta al rigidbody |
| freezeRotation | Controlas si la física afectará al cambio de rotación del objeto |
| constraints | Controla que grados de libertad son permitidos en la simulación del ese Rigidbody |
| collisionDetectionMode | El modo de detección de las colisiones |
| centerOfMass | El centro de masa relativo al transform original |
| worldCenterOfMass | El centro de masa del rigidbody en el world space |
| inertiaTensorRotation | La rotación del tensor de inercia |
| inertiaTensor | El tensor diagonal de inercia de masa relativa para el centro de masa |
| detectCollisions | Está habilitada la detección de colisiones? |
| useConeFriction | La fricción de la Fuerza de cono que se utilizará para este rigidbody |
| position | La posición del rigidbody |
| rotation | La rotación del rigidbody |
| interpolation | Te permite suavizar el efecto de la física corriendo a una velocidad fija |
| solverIterationCount | Le permite anular el número de iteraciones solucionadas por rigidbody |
| sleepVelocity | La velocidad lineal, por debajo del cual los objetos comienzan a dormir |
| sleepAngularVelocity | La velocidad angular, por debajo la cual los objetos comienzan a dormir |
| maxAngularVelocity | La máxima velocidad angular de un rigidbody |



Las funciones son:

| | |
|--------------------------|--|
| SetDensity | Establece la masa sobre la base de los colliders adjunta asumiendo una densidad constante. |
| AddForce | Añade fuerza a el rigidbody |
| AddRelativeForce | Añade una fuerza relativa a las coordenadas del sistema del rigidbody |
| AddTorque | Añade un torque al rigidbody |
| AddRelativeTorque | añade un torque al rigidbody relativo a las coordenadas del sistema |
| AddForceAtPosition | Añade una fuerza en una posición |
| AddExplosionForce | Simula una explosión y aplica la fuerza correspondiente al rigidbody |
| ClosestPointOnBounds | El punto más cercano a la caja de detección de colisiones |
| GetRelativePointVelocity | La velocidad del rigidbody relativa al punto relativePoint |
| GetPointVelocity | La velocidad de un rigidbody en un punto del world space |
| MovePosition | Mueve el rigidbody a position |
| MoveRotation | Rota el rigidbody a rotation |
| Sleep | Fuerza al rigidbody a dormir por lo menos un frame |
| IsSleeping | Esta durmiendo? |
| WakeUp | Fuerza al rigidbody a despertar |
| SweepTest | Comprueba si rigidbody a colisionado con alguien |
| SweepTestAll | Como el anterior, pero devuelve todos los golpes. |

Esta clase tiene las funciones OnCollisionEnter, OnCollisionExit y OnCollisionStay, para enviar mensajes a otros objetos sobre las colisiones que hayan tenido.

vi) Transform (hereda de Component y IEnumerable):

Todo objeto en una escena tiene un componente Transform. Es usado para almacenar y manipular la posición, rotación y escala de los objetos. Todo Transform puede tener un parent (padre), que te permite cambiar y aplicar la posición, rotación y la escala jerárquicamente, en el panel de Jerarquía (Hierarchy).



Las variables son:

| | |
|--------------------|--|
| position | La posición del transform en el world space |
| localPosition | La posición del transform en el local space |
| eulerAngles | La rotación en grados Euler |
| localEulerAngles | La rotación en grados Euler relativos a la rotación del padre |
| right | El eje rojo en el transform world space |
| up | El eje verde en el transform world space |
| forward | El eje azul en el transform world space |
| rotation | La rotación del el transform en el world space almacenada como un Quaternion |
| localRotation | La rotación del transform relativa a la rotación del padre |
| localScale | La escala del transform relativa al padre |
| parent | El padre del transform |
| worldToLocalMatrix | Matriz que transforma un punto del world space dentro de local space |
| localToWorldMatrix | Inverso que la anterior |
| root | Devuelve transform raíz de los transform que hay en Hierarchy |
| childCount | El número de hijos de transform que hay. |
| lossyScale | La scala global del objeto |

Las funciones son:

| | |
|---------------------------|---|
| Translate | Mueve el transform en dirección y distancia de traslación |
| Rotate | Aplica la rotación |
| RotateAround | Rota alrededor de un punto de coordenadas |
| LookAt | Rota el transform de modo que el vector apunte hacia adelante a un objeto |
| TransformDirection | Transforma la dirección del local space a world space |
| InverseTransformDirection | Inversa que la función anterior |
| TransformPoint | Transforma la posición del local space a world space |
| InverseTransformPoint | La inversa que la función anterior |
| DetachChildren | Quitar el padre a todos los hijos |
| Find | Encuentra un hijo por nombre y lo devuelve |
| IsChildOf | Es este transform hijo del padre? |



b) GameObject (hereda de Object):

Esta clase es la clase plantilla para todas las entidades de las escenas en Unity.

Las variables son casi las mismas que la clase component:

| | |
|-----------------|--|
| isStatic | Especifica si el game Object es estatico o no |
| transform | El Transform que está unido al GameObject (null si no hay unido) |
| rigidbody | El Rigidbody que está unido al GameObject (null si no hay unido) |
| camera | El Camera que está unido al GameObject (null si no hay unido) |
| light | El Light que está unido al GameObject (null si no hay unido) |
| animation | El Animation que está unido al GameObject (null si no hay unido) |
| constantForce | El ConstantForce que está unido al GameObject (null si no hay unido) |
| renderer | El Renderer que está unido al GameObject (null si no hay unido) |
| audio | El AudioSource que está unido al GameObject (null si no hay unido) |
| guiText | El GUIText que está unido al GameObject (null si no hay unido) |
| networkView | El NetworkView que está unido al GameObject (null si no hay unido) |
| guiTexture | El GUITexture que está unido al GameObject (null si no hay unido) |
| collider | El Collider que está unido al GameObject (null si no hay unido) |
| hingeJoint | El HingeJoint que está unido al GameObject (null si no hay unido) |
| particleEmitter | El ParticleEmitter que está unido al GameObject (null si no hay unido) |
| gameObject | El game object al que están unidos los componentes |
| tag | El tag del game object. |



El constructor de la clase es `GameObject`, que crea un nuevo objeto llamado `name`(nombre pasado por parámetro).

Las funciones de la clase:

| | |
|-------------------------------------|--|
| <code>GetComponent</code> | Devuelve el componente del tipo o del nombre pasado por parámetro si el objeto tiene unido alguno, si no, devuelve null |
| <code>GetComponentInChildren</code> | Devuelve el componente del tipo seleccionado del <code>gameObject</code> o de alguno de sus hijos usando la búsqueda en profundidad |
| <code>GetComponentInChildren</code> | Devuelve los componentes del tipo seleccionado del <code>GameObject</code> o alguno de sus hijos |
| <code>GetComponents</code> | Devuelve todos los componentes del tipo seleccionado que tiene el <code>GameObject</code> |
| <code>SetActiveRecursively</code> | Cambia el estado a activo de todos los hijos del <code>gameObject</code> |
| <code>CompareTag</code> | Este <code>GameObject</code> está etiquetado como tag? |
| <code>SendMessageUpwards</code> | Invoca al método llamado <code>methodName</code> en cualquier script de ese <code>gameObject</code> y en todos los ancestros de el <code>gameObject</code> |
| <code>SendMessage</code> | Invoca al método llamado <code>methodName</code> en cualquier script de ese <code>gameObject</code> |
| <code>BroadcastMessage</code> | Invoca al método llamado <code>methodName</code> en cualquier script de ese <code>gameObject</code> y todos sus hijos. |
| <code>AddComponent</code> | Añade un componente de la clase <code>className</code> al <code>game object</code> |
| <code>SampleAnimation</code> | Muestra una animación en un momento dado de las propiedades de animación. |



Funciones de la clase:

| | |
|------------------------|---|
| CreatePrimitive | Crea un game object con una textura de renderizado y su collider asociado |
| FindWithTag | Devuelve el GameObject activo etiquetados con tag. |
| FindGameObjectsWithTag | Devuelve una lista de GameObjects etiquetados con tag |
| Find | Busca un game object por nombre (name) |

c) Texture (hereda de Object):

Es la clase base para el manejo de las texturas. Contiene funcionalidades que son comunes en las clases Texture2D y RenderTexture.

Las variables de la clase son:

| | |
|------------|---|
| width | La anchura de la textura en pixeles |
| height | La altura de la textura en pixeles |
| filterMode | El modo de filtro aplicado a la textura |
| anisoLevel | Nivel de Anisotropic filtro para la textura |
| wrapMode | Envuelve el tipo de la textura (Repeat o Clamp) |
| mipMapBias | Mip map bias de la textura |

La clase contiene la función GetNativeTextureID que recupera el "hardware" nativo para manejar una textura.

7) Ray (estructura):

Es la representación de los rays. Un ray es una línea infinita que comienza en origen (origen) y va en cualquier dirección

Las variables son origen (el punto de origen del ray) y direction (la dirección del ray).

El constructor es Ray, crea un ray que comienza en origen hacia direction.



Las funciones son GetPoint, que devuelve el punto en las unidades de distancia a lo largo del rayo, y ToString que devuelve una cadena (string) sobre el rayo con un formato entendible.

8) RaycastHit (estructura):

Es una estructura usada para obtener información sobre un raycast.

Las variables son las siguientes

| | |
|-----------------------|--|
| point | El punto de impacto sobre el world space donde el ray golpea al collider |
| normal | La normal que sobre sale del golpe del ray |
| barycentricCoordinate | La coordenada barycentric del triangulo que golpea |
| distance | La distancia del origen al punto de impacto |
| triangleIndex | El índice del triangulo que ha sido golpeado |
| textureCoord | La textura uv que coordina el punto de impacto |
| textureCoord2 | La textura uv secundaria que coordina el punto de impacto |
| lightmapCoord | El uv lightmap que coordina el punto de impacto |
| collider | El collider que ha sido golpeado |
| rigidbody | El Rigidbody del collider que ha sido golpeado |
| transform | El transform del rigidbody o el collider que ha sido golpeado |

9) Screen:

Es usada para obtener información sobre la pantalla, una lista de resoluciones soportadas, la resolución actual, etc.

Las variables de la clase son:

| | |
|-------------------|--|
| resolutions | Todas las resoluciones soportadas por el monitor |
| currentResolution | La actual resolución |
| showCursor | Mostramos el cursor? |
| lockCursor | Bloqueamos el cursor? |
| width | La anchura actual de la pantalla en píxeles |
| height | La altura actual de la pantalla en píxeles |
| fullScreen | Esta el juego utilizando toda la pantalla? |
| orientation | Especifica la orientación lógica de la pantalla |



La función de la clase es SetResolution con la cual puedes cambiar de resolución a tu juego.

10) Time:

El interfaz para obtener información sobre el tiempo en Unity.

Las variables de la clase:

| | |
|----------------------|---|
| time | El tiempo en el cual el frame ha comenzado. Se cuenta en segundos desde que el juego ha comenzado |
| timeSinceLevelLoad | El tiempo en el cual el frame ha comenzado. Se cuenta desde que el ultimo nivel ha sido cargado |
| deltaTime | El tiempo en segundos que tarda en completar el ultimo frame |
| fixedTime | El tiempo en el cual el último FixedUpdate ha empezado |
| fixedDeltaTime | El intervalo de tiempo en el cual physics y otros fixed frame rate updates se llevan a cabo |
| maximunDeltaTime | El tiempo máximo que un frame puede tomar |
| smoothDeltaTime | A smoothed out Time.deltaTime |
| timeScale | La escala en la que se pasa el tiempo |
| frameCount | EL número total de frames que han sido pasados |
| realtimeSinceStartup | El tiempo real en segundos desde que el juego ha comenzado |
| captureFramerate | Si captureFramerate es inicializado con una valor mayor que 0 el tiempo avanza en x. |

11) Touch (estructura):

Es una estructura en la que se describe el estado de el toque de una dedo en la pantalla.

Las variables son:

| | |
|---------------|---|
| fingerId | El identificador único del touch |
| position | La posición del touch |
| deltaPosition | La posición delta desde el último cambio |
| deltaTime | Acumulación de tiempo pasado desde el último cambio |
| tapCount | Numero de taps |
| phase | Describe la fase del touch |



12) Vector3 (estructura):

Es la representación de vectores 3D y puntos.

Esta estructura es usada para pasar la posición 3D y la dirección.

Las variables:

| | |
|-----------------|---|
| x | El componente x del vector |
| y | El componente y del vector |
| z | El componente z del vector |
| this[int index] | Accede a los componentes x, y, z utilizando [0], [1], [2] respectivamente |
| normalized | Devuelve el vector con magnitud de 1 |
| magnitude | Devuelve la extensión del vector |
| sqrMagnitud | Devuelve la longitud al cuadrado de este vector |

El constructor de la clase es Vector3, crea un nuevo vector con las componentes x, y, z.

Las funciones son:

| | |
|-----------|---|
| Scale | Multiplica todos los componentes por el mismo componentes de escala |
| Normalize | Crea un vector con magnitud 1 |
| ToString | Devuelve una cadena sobre el vector |

Las variables de la clase:

| | |
|---------|---------------------------------|
| zero | Crea el vector Vector3(0, 0, 0) |
| one | Crea el vector Vector3(1, 1, 1) |
| forward | Crea el vector Vector3(0, 0, 1) |
| up | Crea el vector Vector3(0, 1, 0) |
| right | Crea el vector Vector3(1, 0, 0) |

La clase contiene otras funciones que se utilizan para operar entre vectores, como en nuestro proyecto no hemos utilizado estas funciones no las vamos a explicar.



- 3.2) Método de compilación y ejecución de la aplicación de Unity 3d en Android

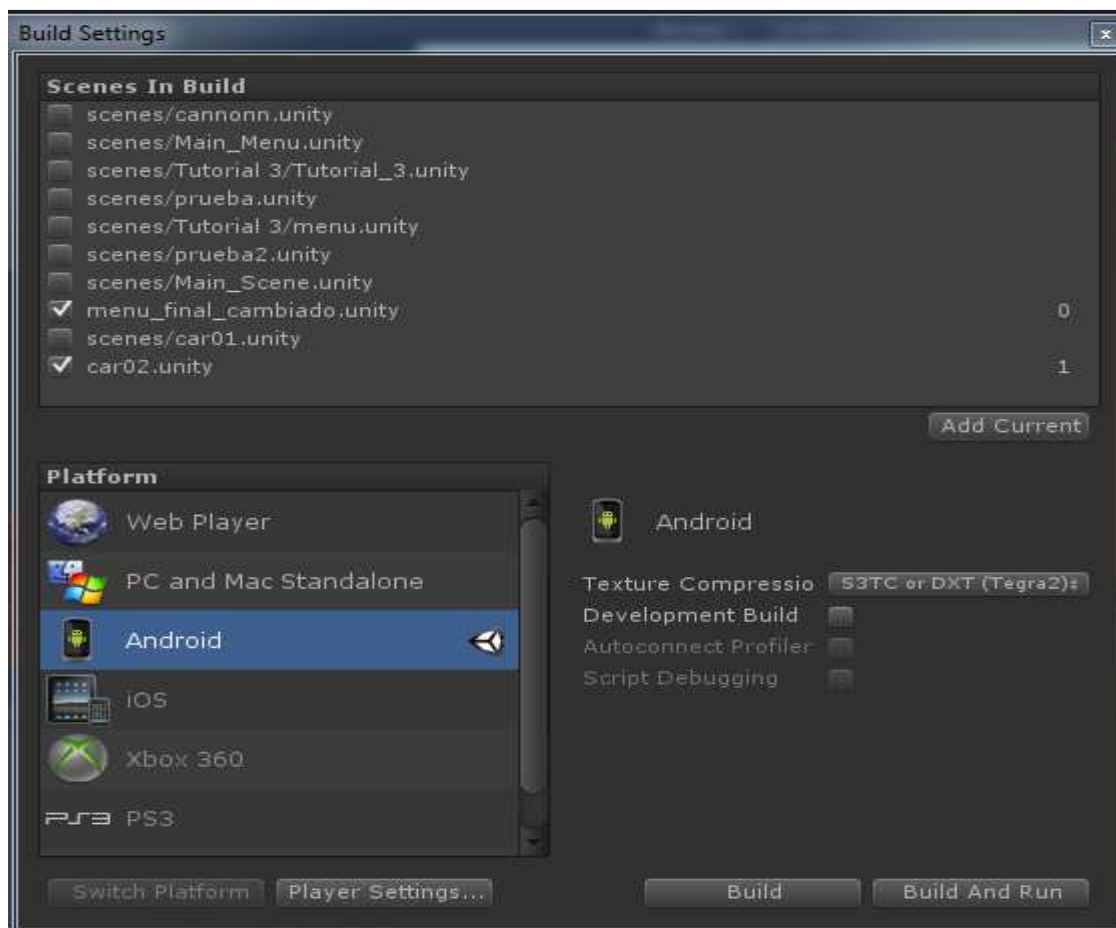
Una vez vistas y explicadas cada una de las clases utilizadas en la aplicación, vamos a explicar el proceso a seguir para que nuestra aplicación corra en un dispositivo con android.

Lo primero que vamos a necesitar es una licencia de Unity 3d para Android, ya que sino no podremos publicar las aplicaciones que hagamos.

Una vez tengamos el proyecto listo para compilarlo (todas las escenas que componen el juego ya están listas para ser compiladas correctamente) tenemos que descargar el SDK de Android y proceder a su instalación.

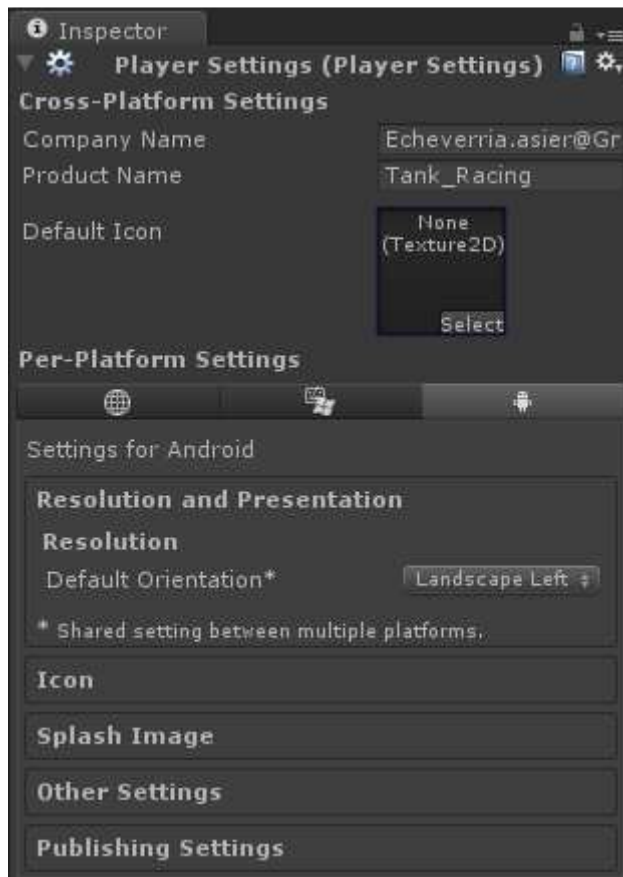
Una vez esté instalado el SDK de Android ya tenemos todo lo necesario para poder compilar la aplicación para Android.

En Unity vamos a la pestaña File y le damos a "build settings" y nos aparecerá la siguiente pantalla





Como se puede apreciar, disponemos de varias plataformas posibles para elegir a la hora de compilar, en mi caso está claro que elegiré Android y ahora le doy al botón de player settings para configurar los diferentes parámetros de mi aplicación en Android. Al darle a este botón vemos la siguiente imagen:

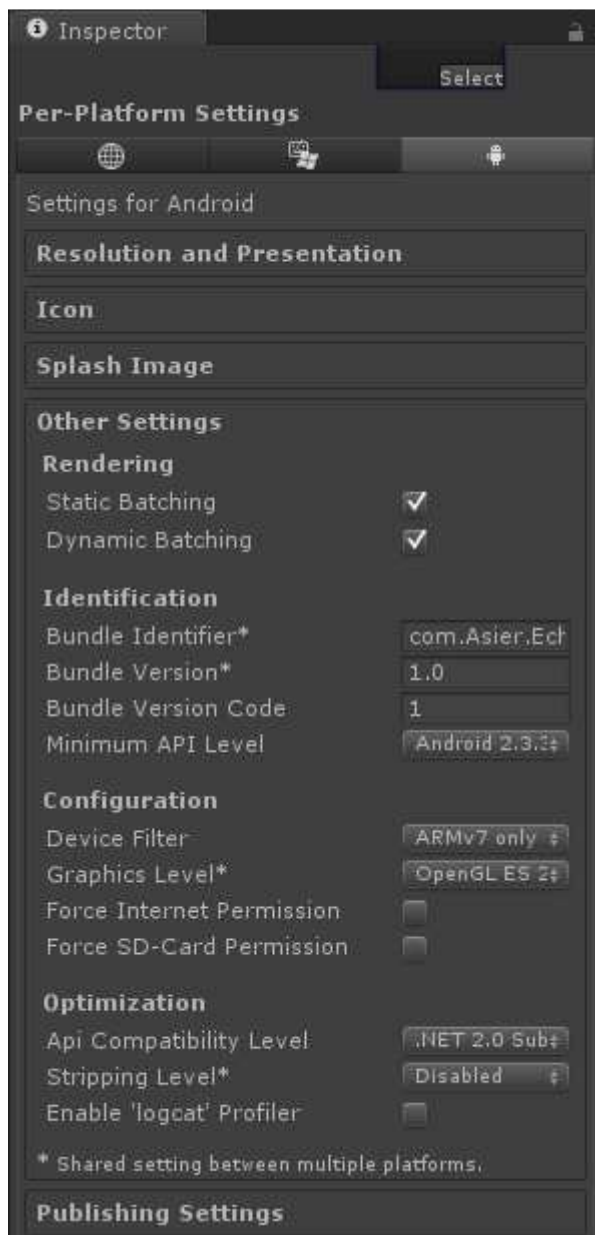


Como vemos tenemos que poner el nombre de la compañía, el título de la aplicación y un icono si queremos poner una imagen en particular, sino Unity pone la que trae por defecto.

Después tenemos que elegir la orientación que va a tener nuestra aplicación en Android y puede ser vertical, horizontal o auto rotación de manera que cuando gires el móvil se cambie de vertical a horizontal y viceversa. En mi caso el juego se aprecia mucho mejor en horizontal, de manera que lo pondré como horizontal.



La otra parte importante que tenemos que configurar es la de other settings que vemos en la siguiente imagen:

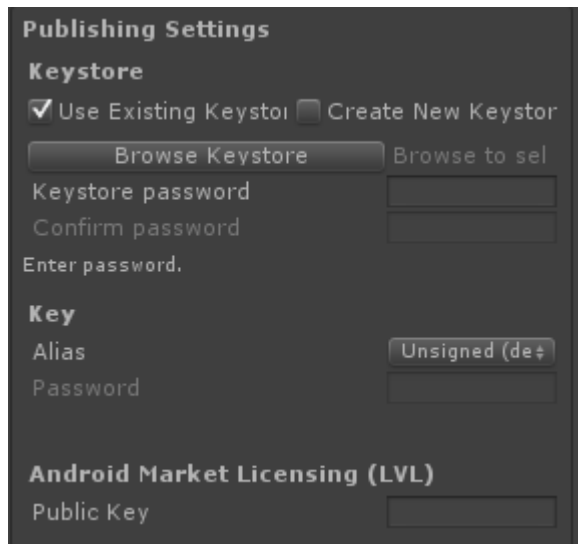


En bundle identifier debemos especificar el nombre del paquete, siempre es recomendable dejar el com. De manera que el nombre del paquete sería com.nombredeempresa.producto

Ahora hay que elegir la API mínima para Android, si ponemos una inferior a 2.2 no nos servirá, hay que elegir una superior, como podemos ver yo he elegido 2.3



La opción de Device Filter nos permite elegir entre tres opciones pero nos quedamos con la primera puesto que las otras son para pasar la aplicación a un emulador y la primera "ARMv7 only" es para pasar la aplicación directamente al móvil.
Por último en publishing settings vemos la siguiente pantalla:



Aquí simplemente tenemos que poner nuestra licencia de Android para que la aplicación se pueda publicar.

Ahora está todo listo para compilar la aplicación de manera que le damos a build y elegimos la carpeta de destino donde se va a guardar nuestro archivo .apk que es la extensión de las aplicaciones para Android.

En mitad de la compilación se nos pedirá indicar el directorio donde instalamos anteriormente el SDK de Android para que use las librerías y archivos necesarios para compilar correctamente la aplicación.



Una vez le hayamos indicado donde tenemos instalado el SDK de Android la compilación terminará correctamente si no tenemos fallos en nuestra aplicación y estará lista para pasarla al móvil y proceder a instalarla y usarla.



3.3.Análisis:

- **Análisis de requisitos**

A continuación vamos a ver los requisitos propios de cada escena de nuestro juego. El juego constará de dos escenas, una será la parte del menú propiamente dicha y otra es la escena en la que transcurrirá el juego en si.

Escena del menú:

- Se dividirá en dos partes, un primer menú que constará de las opciones de jugar y salir de la aplicación, y el otro menú que se llegará a él dándole al botón de jugar que constará de el botón de 1 jugador, el botón controles y el botón atrás. Dándole al botón de 1 jugador pasaremos a la segunda escena, en controles veremos como se juega y el botón atrás nos permite volver al menú anterior.
- La forma de pulsar los botones será táctil, dando un toque sobre alguno de los botones realizaremos la acción.
- Los menús tendrán efectos para darles una mejor apariencia como veremos posteriormente cuando describamos todos los objetos de esta escena.

Escena del juego:

- El juego consistirá en manejar un tanque cuya misión es realizar un recorrido por el mapa en 3d de una habitación y llegar a la meta en el menor tiempo posible, habiendo destruido el mayor número de objetos posible para obtener una mayor puntuación.
- Para moverse con el tanque usaremos el stick analógico izquierdo y para mover el cañón y apuntar a los objetivos usaremos el stick derecho.
- La escena posee varios botones, uno para cambiar la vista de la cámara y elegir la que mas se ajuste a nuestro gusto para manejar el tanque y apuntar a los objetivos, otro para disparar a los objetivos, otro para pausar el juego, otro para ver los controles por si no sabemos como manejar el tanque y otro para renacer por si nos hemos desviado o quedado atrancados podamos volver al último punto que el juego detectó que habíamos pasado.
- También dispondremos de un temporizador y de un marcador que indique el tiempo restante y los puntos que llevamos respectivamente.



- Por último al acabarse el tiempo nos saldrá un menú diciendo que nuestro tiempo se ha agotado y dos botones, uno para salir del juego y otro para volver al menú principal.
- Lo mismo si llegamos a tiempo a la meta, nos saldrá un menú indicando la puntuación obtenida y los dos botones descritos en el punto anterior.

Diagramas de estado de la aplicación:

Diagrama de estado de la escena del menú principal:

Diagrama del menú principal con las diferentes opciones posibles para hacer.

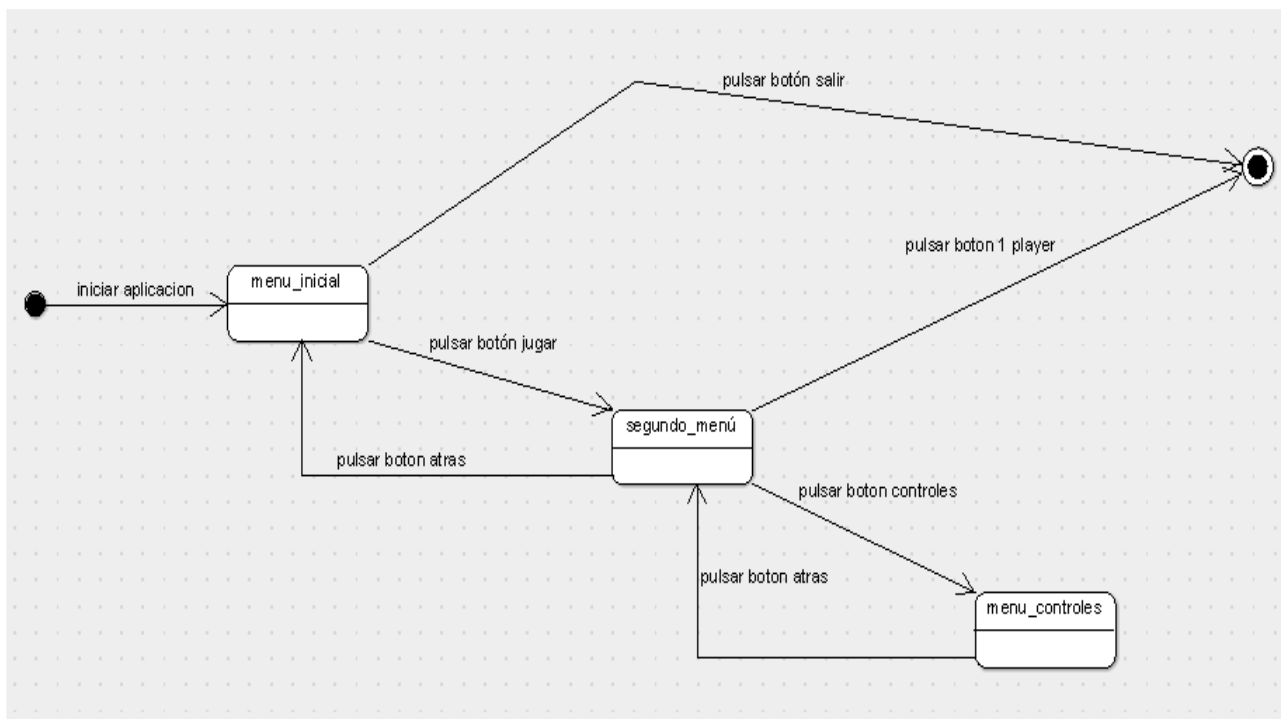
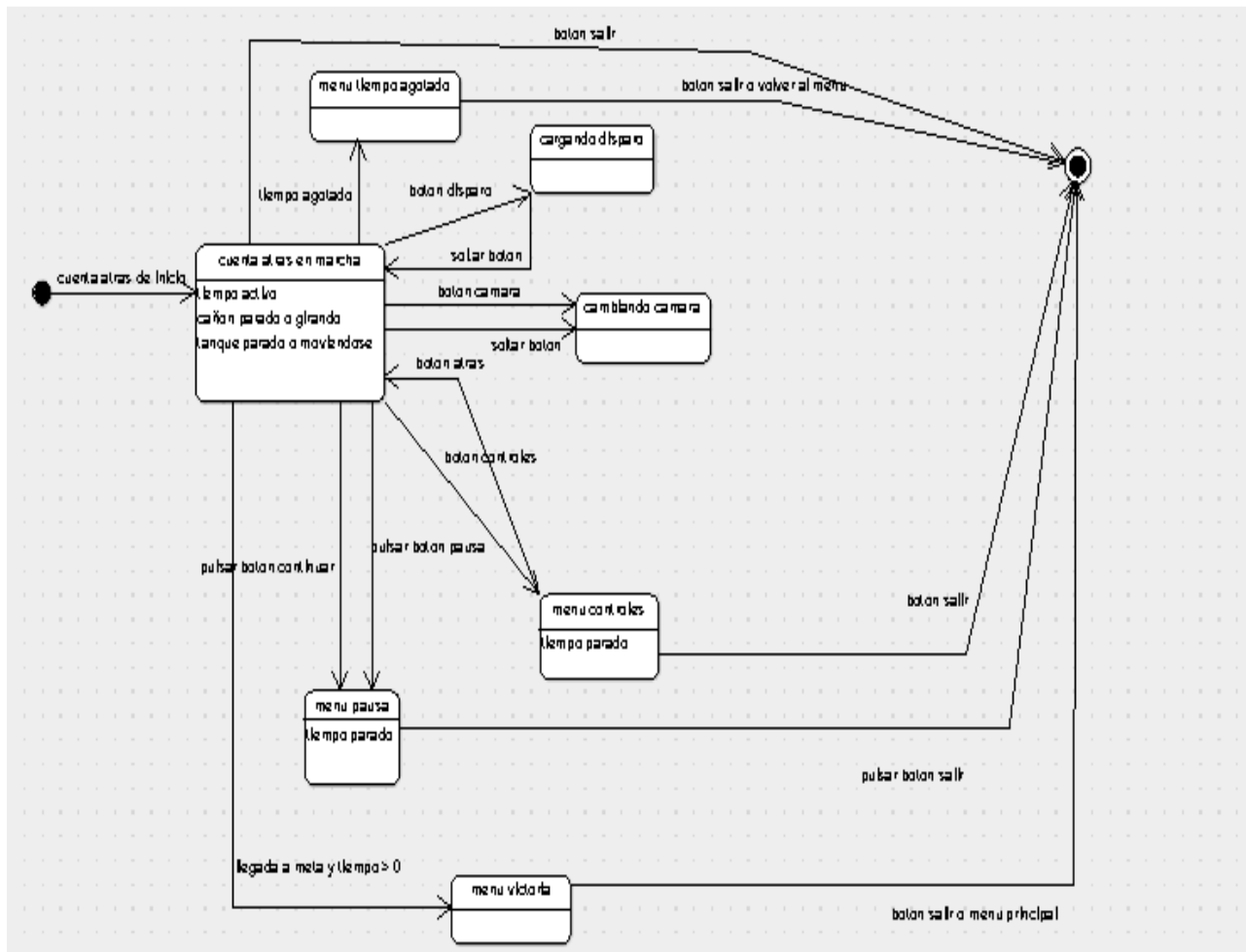




Diagrama de estado de la escena de la escena del juego:

Diagrama de la escena del juego con las opciones posibles a realizar y los estados a los que llegamos al pulsar cada botón.





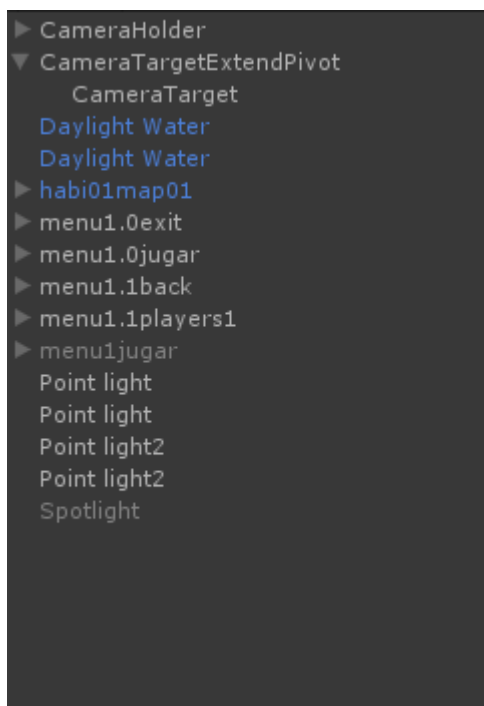
3.4. Diseño e Implementación:

En este apartado vamos a ir explicando cómo hemos realizado la implementación de nuestro proyecto por escenas.

Escena del Menú:

A continuación explicaremos los diversos objetos que forman la escena del menú principal así como sus funciones y características.

Veamos el esquema de objetos inicial de la escena menú "menu":

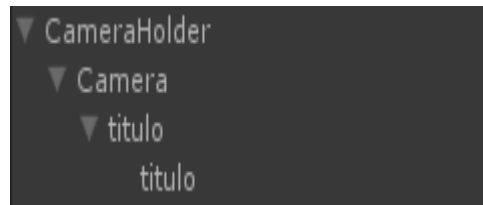


En esta imagen aparecen todos los objetos, ordenados alfabéticamente, que se encuentran en escena. Los objetos pueden destruirse y desaparecer de la lista, o deshabilitarse, en este caso el nombre del objeto tomara un tono gris. Los objetos de



azul de la lista son objetos que están prefabricados y que han sido añadidos a la escena. Esto resulta útil cuando vas a utilizar los mismos objetos en diferentes escenas, para no tener que crear en cada escena un objeto nuevo.

El primer objeto que podemos ver en el historial es CameraHolder, contiene los objetos que se muestra a continuación



El objeto **Camera** es el encargado de mostrar la escena. En una escena siempre debemos tener al menos una cámara para poder ver la escena al darle al empezar el juego.



Con el componente Transform controlamos la posición de la cámara y por consiguiente el ángulo y la perspectiva que queremos ver en nuestro juego. En el componente Camera se puede configurar varios atributos de la cámara, para obtener diferentes resultados. En mi aplicación el valor que he modificado principalmente es "Field of View" con el que configuras la distancia focal, de este modo obtienes que los objetos se vean más o menos lejos a pesar de que se encuentren en la misma posición. Si deseamos, con el componente GUI Layer podemos introducir a la imagen resultante de la escena componentes de la interfaz de Unity.



Otro importante es Audio Listener:

Es el receptor de sonidos de la escena, a través de este componente los sonidos que producen otros objetos serán escuchados. El sonido de Unity está configurado tridimensionalmente, con esto quiero decir que dependiendo donde se encuentre el emisor de sonido (Audio Emitter) y el receptor (Audio Listener), el sonido se escuchará más o menos según la distancia.

Para conseguir que la cámara siga o mire a algo en concreto, hemos utilizado el script MouseOrbit como se puede ver en el dibujo, y lo que hace es mediante este código

```
using UnityEngine;
using System.Collections;

public class MouseOrbit : MonoBehaviour
{
    public Transform target;
    public float distance = 10.0f;

    public float xSpeed = 250.0f;
    public float ySpeed = 120.0f;

    public float yMinLimit = -20;
    public float yMaxLimit = 80;

    public float xMinLimit = -7;
    public float xMaxLimit = 7;

    private float x = 0.0f;
    private float y = 0.0f;

    void Start ()
    {
        var angles = transform.eulerAngles;
        x = angles.y;
        y = angles.x;
    }

    void Update ()
    {
        if (target)
        {
            y = ClampAngle(y, yMinLimit, yMaxLimit);
            x = ClampAngle(x, xMinLimit, xMaxLimit);

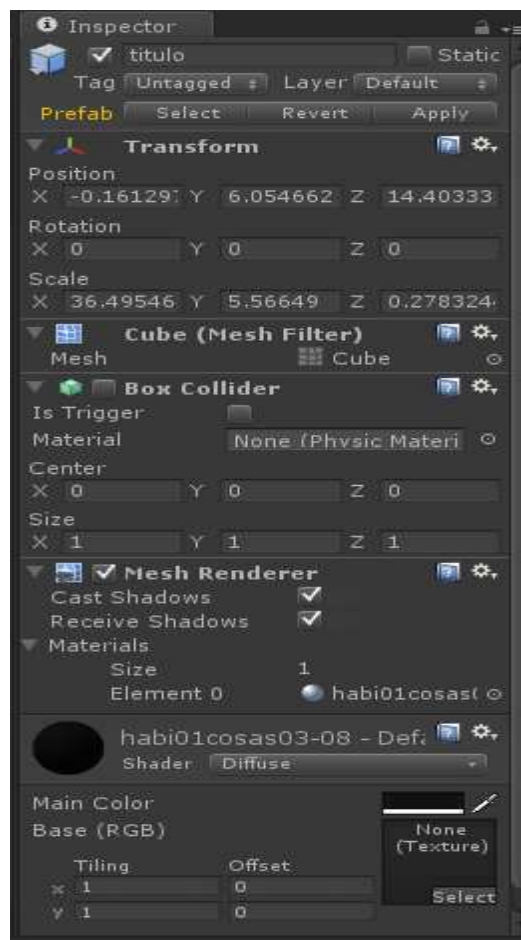
            transform.rotation = Quaternion.Euler(y, x, 0);
            transform.position = (Quaternion.Euler(y, x, 0) * new Vector3(0.0f, 0.0f, -distance) + target.position);
        }
    }

    static float ClampAngle(float angle, float min, float max)
    {
        if (angle < -360)
        {
            angle += 360;
        }
        if (angle > 360)
        {
            angle -= 360;
        }
        return Mathf.Clamp(angle, min, max);
    }
}
```



Con este script le indicamos que siga al objeto que pongamos en la variable target (en mi caso CameraTarget) que le mire desde la distancia 10. Si quisiéramos podríamos indicarle que siga al objeto en cuestión.

El objeto titulo simplemente es un objeto que mostrará el título del juego.



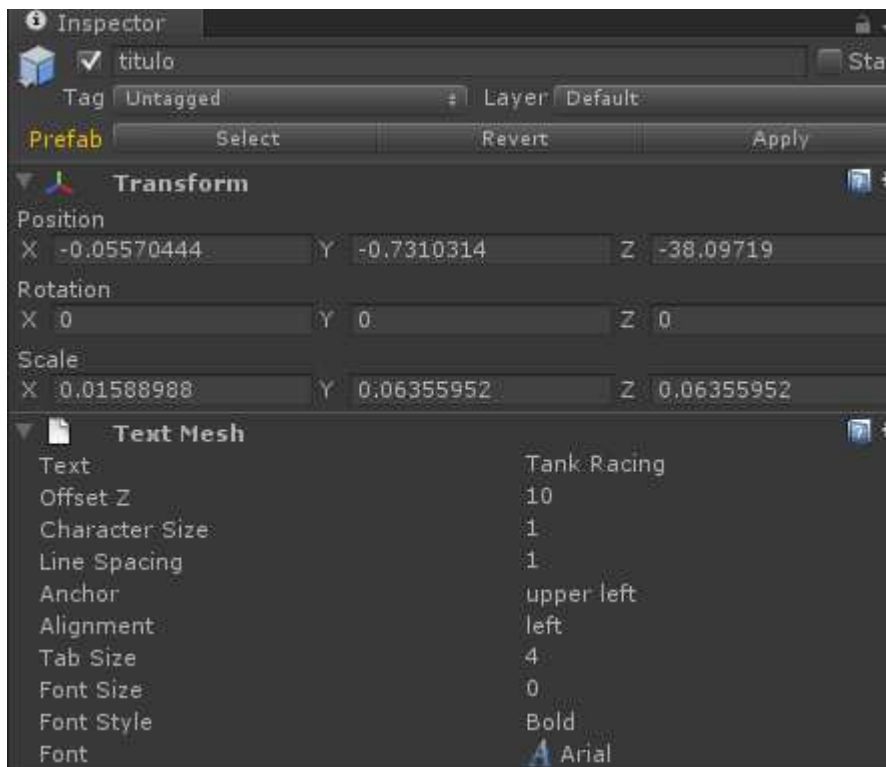
Cuando tienes seleccionado un objeto en la lista de la vista Hierarchy, aparecen todos los componentes que contiene el objeto en la vista Inspector. Como podemos observar en esta imagen, este es la vista Inspector del objeto titulo. Posee el componente Transform con el cual puedes controlar la posición y rotación del objeto en escena.



Box Collider es utilizado para detectar colisiones pero en esta escena este componente no será relevante ya que no vamos a controlar ninguna colisión de objetos contra la pared. Como se puede observar en el componente Mesh Renderer, la pared está configurada para recibir sombras y vemos que tiene asignada una textura.

Cuando se asigna una textura a un objeto, se crea un directorio en la carpeta raíz de la escena con los materiales de los objetos y se crea un material con la textura asignada.

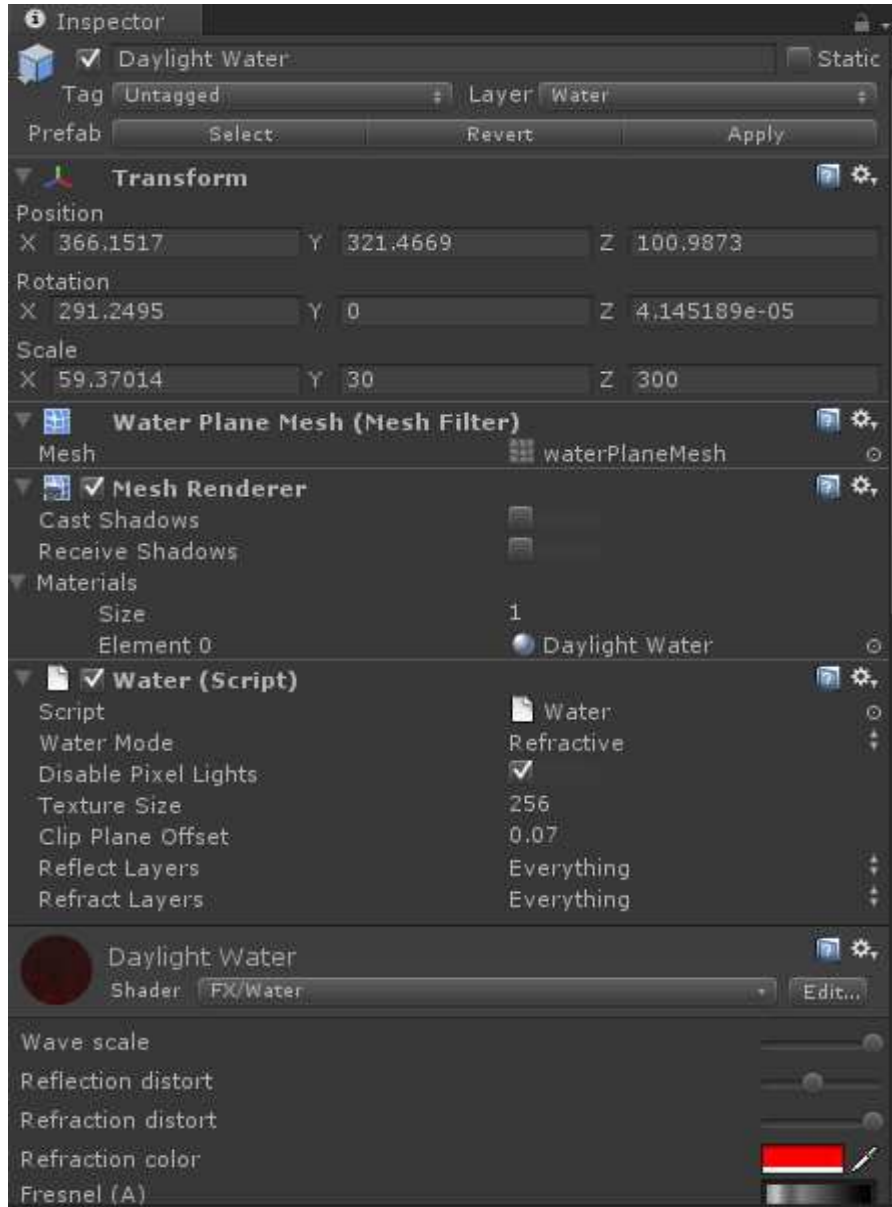
Como vemos disponemos de otro objeto dentro de titulo con el mismo nombre, y este lo que hace es escribir el texto del titulo ya que el anterior sólo era un recuadro con una textura en donde va el título. Como vemos en Text Mesh indicamos el tipo de letra que vamos a poner y el tamaño deseado.



El siguiente objeto en la lista después de Camera Holder es CameraTargetExtendedPivot y contiene al objeto CameraTarget que es a lo que va a apuntar nuestro objeto Camera, como podemos ver en la imagen correspondiente a la cámara que hemos explicado anteriormente.



Los dos siguientes elementos se llaman Daylight Water y su uso simplemente es para dar un efecto relativo a las ondas de agua al menú.



Como se puede observar el Shader ya viene con Unity 3D y se pueden configurar la escala de las ondas o la refracción entre otros parámetros. A continuación vamos a ver la diferencia que hay entre ver el menú con estos dos objetos deshabilitados y después con los objetos habilitados para ver la diferencia que hay.



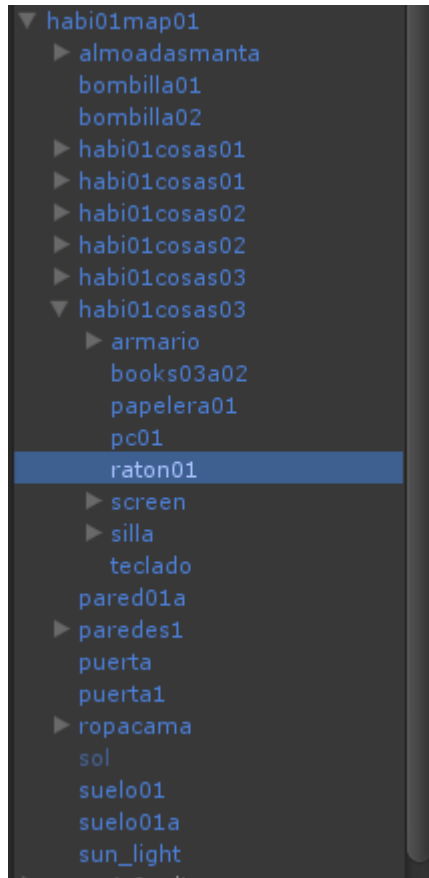
Como vemos los botones son muy simples así que aplicaremos el efecto sobre el menú y queda de la siguiente manera:



Como podemos ver el efecto es muy notorio y le da un toque bastante curioso al menú de manera que ahora se ve como si fueran ondas de agua y con un efecto transparente que queda muy bien.



El siguiente objeto que vemos es “habi01map01” este es un objeto muy importante en nuestro juego ya que posee todos los objetos que componen el escenario 3D de nuestro juego, como podemos ver hay muchos objetos cada uno sus texturas correspondientes y sus colliders para detectar colisiones.

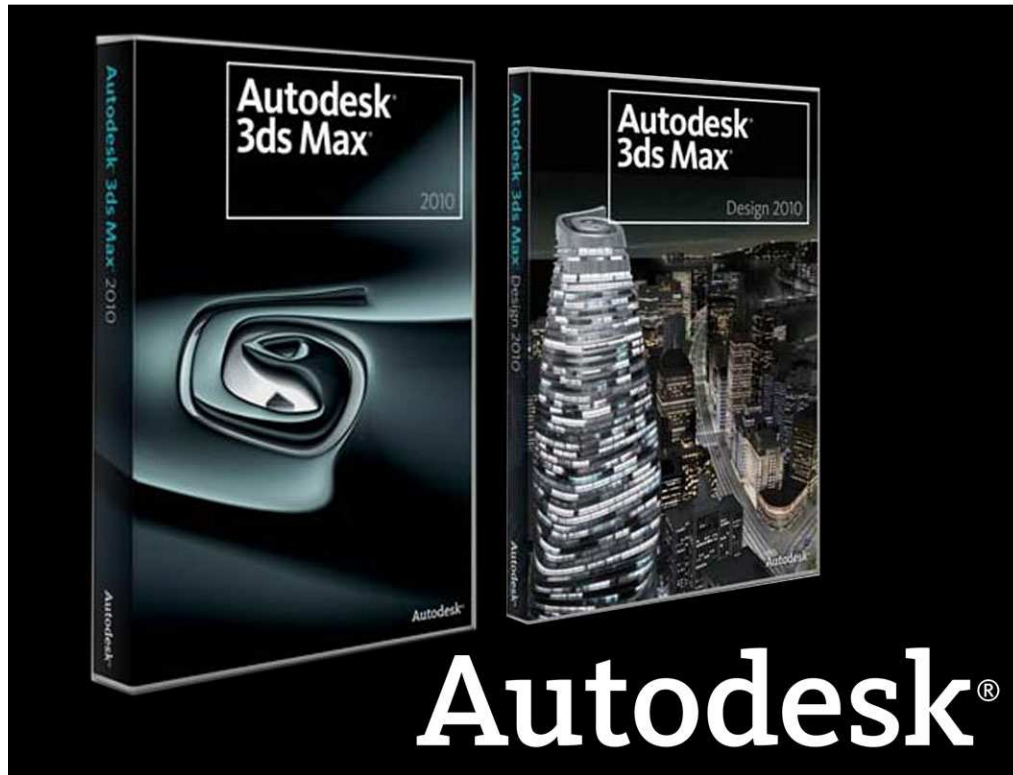


Como podemos ver hay numerosos objetos que forman la habitación al completo, desde el suelo de la habitación, las paredes, una cama etc. La habitación ha sido creada con otros programas orientados al diseño 3D que son **AutoDesk 3ds Max** (antes conocido como **3D Studio Max**) y **Maya**.

Autodesk 3ds Max es un programa de creación de gráficos y animación 3D desarrollado por Autodesk, en concreto por la división Autodesk Media & Entertainment (anteriormente Discreet). Creado inicialmente por el grupo Yost para Autodesk, salió a la venta por primera vez en 1990 para DOS.



3ds Max, con su arquitectura basada en plugins, es uno de los programas de animación 3D más utilizado, especialmente para la creación de video juegos, anuncios de televisión, en arquitectura o en películas.



3ds Max es uno de los programas de animación 3D más utilizados. Dispone de una sólida capacidad de edición, una omnipresente arquitectura de plugins y una larga tradición en plataformas Microsoft Windows. 3ds Max es utilizado en mayor medida por los desarrolladores de videojuegos, aunque también en el desarrollo de proyectos de animación como películas o anuncios de televisión, efectos especiales y en arquitectura.

Desde la primera versión 1.0 hasta la 4.0 el programa pertenecía a Autodesk con el nombre de 3d Studio. Más tarde, Kinetix compró los derechos del programa y lanzó 3 versiones desde la 1.0 hasta la 2.5 bajo el nombre de 3d Studio Max. Más tarde, la empresa Discreet compró los derechos, retomando la familia empezada por Autodesk desde la 4.0 hasta 6.0 también bajo el nombre de 3d Studio Max. Finalmente, Autodesk retomó el programa desarrollándolo desde la versión 7.0 en adelante bajo el mismo nombre, hasta la versión 9. A partir de ésta, se denomina Autodesk 3d Studio Max.



Este programa es uno de los más reconocidos modeladores de 3d masivo, habitualmente orientado al desarrollo de videojuegos, con el que se han hecho enteramente títulos como las sagas 'Tomb Raider', 'Splinter Cell' y una larga lista de títulos de la empresa Ubisoft.

Por otro lado, también he utilizado el programa **Maya**. es un programa informático dedicado al desarrollo de gráficos en 3d, efectos especiales y animación. Surgió a partir de la evolución de Power Animator y de la fusión de Alias y Wavefront, dos empresas canadienses dedicadas a los gráficos generados por ordenador. Más tarde Silicon Graphics (ahora SGI), el gigante informático, absorbió a Alias-Wavefront, que finalmente ha sido absorbida por Autodesk.

Maya se caracteriza por su potencia y las posibilidades de expansión y personalización de su interfaz y herramientas'. MEL (Maya Embedded Language) es el código que forma el núcleo de Maya, y gracias al cual se pueden crear scripts y personalizar el paquete. El programa posee diversas herramientas para modelado, animación, render, simulación de ropa y cabello, dinámicas (simulación de fluidos), etc.

Además **Maya** es el único software de 3D acreditado con un Oscar gracias al enorme impacto que ha tenido en la industria cinematográfica como herramienta de efectos visuales, con un uso muy extendido debido a su gran capacidad de ampliación y personalización.

Maya trabaja con cualquier tipo de superficie NURBS, Polygons y Subdivision Surfaces, incluye la posibilidad de convertir entre todos los tipos de geometría.

- **Nurbs:** Son figuras creadas a base de curvas y superficies cuyos componentes son básicamente los CV's (control vertex), las isoparms (isoparamétricas) y los hulls (loops enteros de isoparms).
- **Polygons:** Son los objetos más fáciles de modelar por su falta de complejidad y su mayor número de herramientas. Sus componentes básicas son las Faces (caras), Edges (aristas) y Vertex (vértices).
- **Subdivisiones:** Son un híbrido entre las Nurbs y los Polygons. Sin embargo no se pueden modelar usando ambos estilos a la vez, para ello hay que escoger en qué modo se desea modelar (Standard Mode o Polygon Mode). Poseen los mismos componentes que las Nurbs y los Polygons además de un modo de refinamiento por niveles para obtener mayor subdivisión geométrica y conseguir así mayor detalle de modelado.



Con el **Maya** es con el que he realizado todo el trabajo de diseño 3d, los objetos no los he hecho yo desde 0 ya que este proyecto consistía en elaborar un juego en Unity 3D para Android, así que la mayoría los he bajado y con este programa lo que he hecho es reducir los polígonos aristas y vértices al máximo y he ido juntando los objetos (puertas, ventanas, cama, silla...) hasta terminar de hacer la habitación conforme la quería.

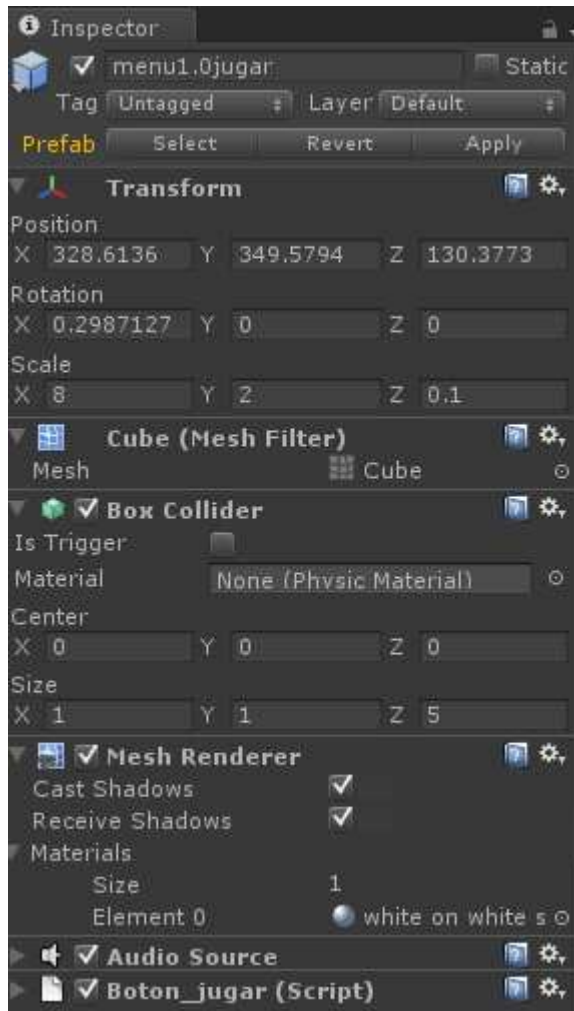
A continuación muestro una serie de imágenes en las que podemos las distintas partes de la habitación.



Como se puede apreciar, se parece bastante a una habitación real y es aquí donde transcurrirá el juego que voy a realizar.



Los 4 siguientes objetos son menu1.0exit, menu1.0jugar y menu1.1back, menu1.1players1. Estos botones en realidad son cajas creadas como objetos básicos de Unity 3D desde GameObject.



Como podemos ver simplemente son cajas con un Box Collider y una malla de cubo a las cuales les hemos añadido funcionalidad mediante sus correspondientes scripts. En este caso el botón jugar tiene el script asociado Boton_jugar.js realizado en JavaScript y como se ve en la imagen que tiene el siguiente código



```
var prefab:GameObject;
var prefab2:GameObject;
var posiX:float;
var posiY:float;
var posiZ:float;
var msg : String;
function Update () {

    for (var i = 0; i < Input.touchCount; ++i) {
        if (Input.GetTouch(i).phase == TouchPhase.Began) {|
            var hit : RaycastHit;
            var ray = Camera.current.ScreenPointToRay (Input.GetTouch(i).position);
            if (Physics.Raycast(ray, hit,Mathf.Infinity)) {|
                if(hit.collider.gameObject == this.gameObject) {|
                    prefab.transform.position.x = posiX;
                    prefab.transform.position.y = posiY;
                    prefab.transform.position.z = posiZ;
                }
            }
        }
    }
}
~
```

Este Script como se puede ver lo que hace es en la función Update comprobar cada frame si algún Touch (cada Touch es cada toque que hagamos en la pantalla del móvil con el dedo) que se registra con Input.GetTouch(i) toca el botón asociado a este Script, en este caso el botón "jugar".

Esto se detecta mediante la creación de un rayo con Physics.Raycast que vaya desde donde se ha detectado el toque en la pantalla táctil hasta el infinito, y si tocara el collider del botón en algún momento moveríamos la cámara al siguiente menú, moviendo las coordenadas de la cámara a las que le marcamos en este Script.

Si quisiéramos hacerlo para que lo reconociera con el ratón en vez de manera táctil simplemente tendríamos que asociar al objeto un Script como este



```
var mouseDown:boolean = false;
var prefab:GameObject;
var prefab2:GameObject;
var posiX:float;
var posiY:float;
var posiZ:float;

function OnMouseDown(){
    mouseDown = true;
}

function Update(){
    if(mouseDown){
        prefab.transform.position.x = posiX;
        prefab.transform.position.y = posiY;
        prefab.transform.position.z = posiZ;
    }

    if((prefab.transform.position.x == posiX) && (prefab.transform.position.y == posiY) && (prefab.transform.position.z == posiZ))
        mouseDown = false;
}
}
```

Básicamente hace lo mismo sólo que ahora comprobamos si se ha detectado un click de mouse en el objeto en vez de un Touch.

Para el botón "Salir" (el objeto menu1.0exit) lo que hay que hacer es asociar otro Script similar al del botón jugar pero variando la acción a emprender cuando detectemos que un Touch a tocado el botón "Salir"



```
var prefab:GameObject;
var prefab2:GameObject;
var posiX:float;
var posiY:float;
var posiZ:float;
var msg : String;
function Update () {

    for (var i = 0; i < Input.touchCount; ++i) {
        if (Input.GetTouch(i).phase == TouchPhase.Began) {
            var hit : RaycastHit;
            var ray = Camera.current.ScreenPointToRay (Input.GetTouch(i).position);
            if (Physics.Raycast(ray, hit,Mathf.Infinity)) {
                if(hit.collider.gameObject == this.gameObject) {
                    Application.Quit();
                }
            }
        }
    }
}
```

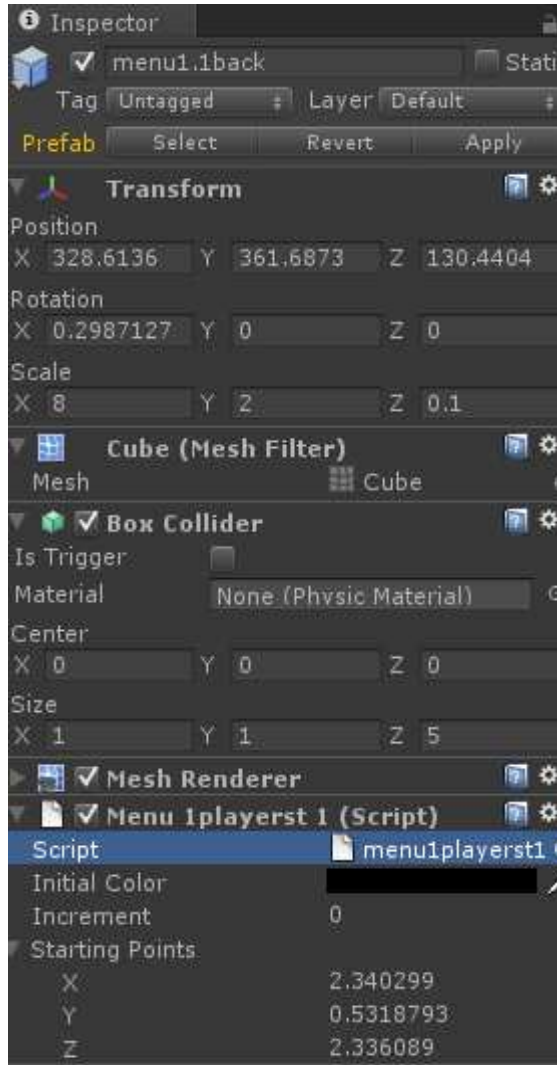
Como se puede ver simplemente hacemos un Application.Quit() ya que lo que se quiere es salir de la aplicación.

Los otros dos botones son otro menú al que llegamos al darle a jugar que es el siguiente menú





Bueno he de decir que cuando se toca uno de los botones este se ilumina y hace un sonido, esto se consigue mediante el siguiente Script





El script menu1playerst1.js también está hecho en JavaScript y contiene el siguiente código

```
var initialColor:Color;
var increment: float = 1;
//var sound1:GameObject;
//var sound2:GameObject;

//Record the starting location of the cube
var startingPoints: Vector3;
startingPoints = Vector3(transform.position.x, transform.position.y, transform.position.z);
function Start()
{
    initialColor = renderer.material.color;
}
//OnMouseEnter is called when the mouse entered the GUIElement or Collider
function OnMouseEnter(){
    Debug.Log("Mouse Enter");
    audio.Play();
}
function OnMouseExit(){
    Debug.Log("Mouse Exit");
    transform.position = startingPoints; //move the cube back to where it started
    renderer.material.color = initialColor;
    increment = 0 ;
}
//OnMouseOver is called every frame while the mouse is over the GUIElement or Collider
function OnMouseOver(){
    Debug.Log("Mouse Over");
    increment = increment-0.0006;
    renderer.material.color = Color.red;
    if(transform.position.z < 1.5)
        transform.position.z = 1.5;
}
}
```

Como se puede apreciar si pasamos el ratón por encima el botón se ilumina simplemente se hace cambiando la intensidad del color, y se aplica un sonido mediante el comando audio.Play(). **Mirarlo para móviles táctiles.**



El botón de 1 player sigue el mismo funcionamiento que el resto de los botones y su Script es

```
function Update () {  
  
    for (var i = 0; i < Input.touchCount; ++i) {  
        if (Input.GetTouch(i).phase == TouchPhase.Began) {  
            var hit : RaycastHit;  
            var ray = Camera.current.ScreenPointToRay (Input.GetTouch(i).position);  
            if (Physics.Raycast(ray, hit,Mathf.Infinity)) {  
                if(hit.collider.gameObject == this.gameObject) {  
                    if ((map == 1))      {  
                        l=1;  
                    }  
                }  
            }  
        }  
    }  
}
```



Cuando se detecte que un dedo a tocado el botón de 1 player se pone la variable l a 1 y eso lo que hace es que se ejecute el siguiente código

```
var map : int;
var mySkin2 : GUISkin;
var f=0;
var l=0;
function game_over() {
    yield WaitForSeconds(2);
}
function empieza_game() {
    yield WaitForSeconds(1);
    f=1;
}
function empieza_game2() {
    yield WaitForSeconds(1);
    f=2;
}
function empieza_game3() {
    yield WaitForSeconds(1);
    f=3;
}
function empieza_game4() {
    yield WaitForSeconds(1);
    f=4;
}
function empieza_game5() {
    f=6;
    Application.LoadLevel(1);
}
function OnGUI() {
    if(l==1) {
        GUI.skin=mySkin2;
        if(f==0) {
            GUI.Box(Rect(0,0,Screen.width,Screen.height),"STARTING GAME IN");
            empieza_game();
        }
        if(f==1) {
            GUI.Box(Rect(0,0,Screen.width,Screen.height),"3");
            empieza_game2();
        }
        if(f==2) {
            GUI.Box(Rect(0,0,Screen.width,Screen.height),"2");
            empieza_game3();
        }
        if(f==3) {
            GUI.Box(Rect(0,0,Screen.width,Screen.height),"1");
            empieza_game4();
        }
        if(f==4) {
            GUI.Box(Rect(0,0,Screen.width,Screen.height),"GO!");
            empieza_game5();
        }
        if(f==6) {
            GUI.Box(Rect(0,0,Screen.width,Screen.height),"GO!");
        }
    }
}
```




Como vemos cuando `l` vale 1 empezamos a ejecutar todo el código que hay en la función `OnGUI`. Esto lo hago porque quiero hacer una pequeña cuenta atrás para que no pille por sorpresa al usuario el comienzo de la partida. De tal manera que `f` va pasando de los valores 0 a 6 y se va imprimiendo por pantalla las diferentes frases que se ven en los bucles `If`.

Se ve la pantalla negra y las letras en medio rojas gracias a que hemos usado un `Skin` personalizado como vemos arriba la variable `var mySkin2 : GUISkin`. Simplemente se le asigna el `background color` a negro y las letras a rojo y en el medio de la pantalla. El botón de atrás simplemente tiene un `Script` asociado que hace que volvamos al menú anterior, moviendo la cámara a las coordenadas deseadas.

```
var prefab:GameObject;
var prefab2:GameObject;
var posiX:float;
var posiY:float;
var posiZ:float;
var msg : String;
function Update () {

    for (var i = 0; i < Input.touchCount; ++i) {
        if (Input.GetTouch(i).phase == TouchPhase.Began) {
            var hit : RaycastHit;
            var ray = Camera.current.ScreenPointToRay (Input.GetTouch(i).position);
            if (Physics.Raycast(ray, hit,Mathf.Infinity)) {
                if(hit.collider.gameObject == this.gameObject) {
                    prefab.transform.position.x = posiX;
                    prefab.transform.position.y = posiY;
                    prefab.transform.position.z = posiZ;
                }
            }
        }
    }
}
```

Y por último el botón de controles que explica los controles del juego por si queremos verlos antes de empezar.

Los otros cuatro últimos objetos simplemente corresponden a la creación de objetos para una adecuada iluminación de la escena, estos objetos se crean desde

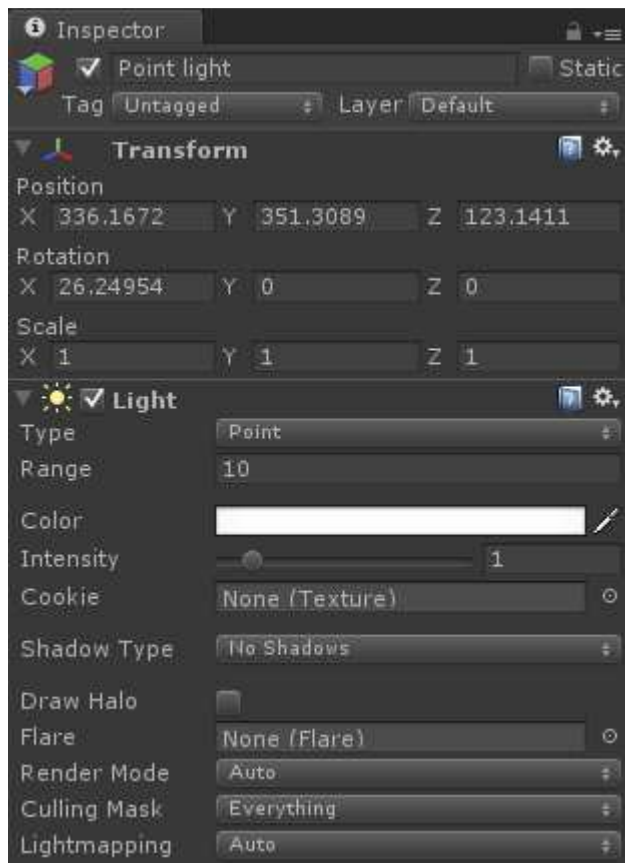
`GameObject -> Create Other -> spotlight` o

`GameObject->Create Other->Point Light`.

En función de cuanto queramos iluminar hay que jugar un poco cambiando los parámetros de intensidad y rango.



En la imagen siguiente podemos ver dónde se cambian estos parámetros:



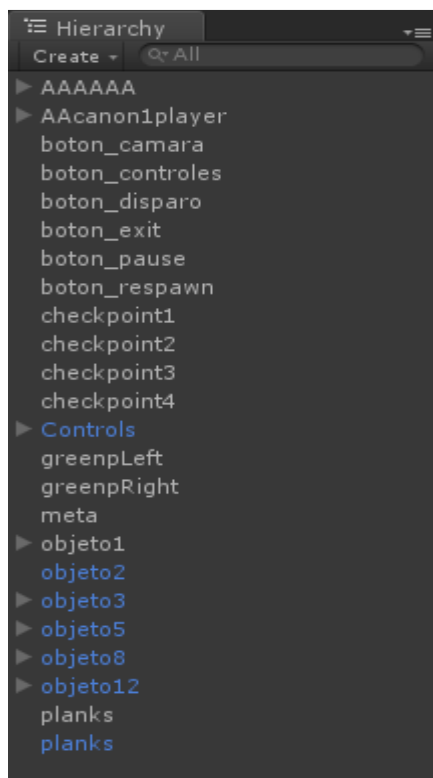


Escena del Juego:

A continuación voy a describir mi otra escena, que es en la que transcurre el juego y tiene más objetos que la del menú ya que es una escena más compleja con más scripts.



En la imagen que podemos ver a continuación mostramos los objetos de la escena en cuestión:





Bien ahora empezaremos a describir los objetos diciendo cual es su función y sus scripts asociados.

Lo primero que vamos a describir son los elementos de la GUI de Unity que consta de una serie de botones y texturas que ahora empezaremos a comentar.



Lo primero que vamos a describir de la GUI de Unity de esta escena son los botones que tenemos en ella.

- Botón de pause: este botón sirve para parar el juego si necesitamos hacerlo mientras estamos jugando por cualquier motivo o interrupción que nos surja.



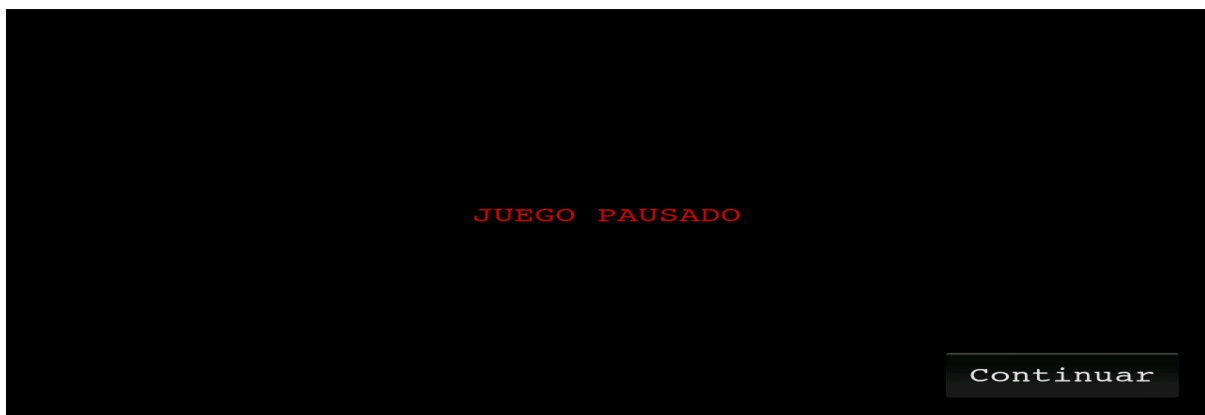
Cuando le demos al botón nos aparecerá una pantalla indicando que el juego está pausado y un botón nuevo "continuar" que servirá para seguir con el juego donde nos habíamos quedado. Esto se consigue al darle al botón de pause mediante el siguiente script asociado a este botón que es el siguiente:

```
var l=0;
var mySkin2 : GUISkin;
function Update () {

    if(Input.touchCount > 0 ){
        for(var i : int = 0; i < Input.touchCount; i++){
            var touch : Touch = Input.GetTouch(i);
            if(touch.phase == TouchPhase.Began && guiTexture.HitTest(touch.position)) {
                l=1;
                Time.timeScale=0;
            }
        }
    }
}

function OnGUI () {
    if(l==1) {
        Time.timeScale=0;
        GUI.skin=mySkin2;
        GUI.Box(Rect (0,0,Screen.width,Screen.height),"JUEGO PAUSADO");
        if (GUI.Button(Rect(1000, 600, 250, 80),"Continuar")) {
            l=0;
            Time.timeScale=1;
        }
    }
}
```

Como se puede apreciar, este script lo que hace es mirar en la función Update, que mira cada frame, si se detecta algún dedo en la pantalla, y si se ha detectado, miramos si este dedo a tocado el botón "pause".





En el caso de que sea así, paramos el tiempo poniendo la variable `Time.timeScale` a cero y ponemos una variable `l` a uno que indica al código de la función `OnGUI` que se ejecute el `if` que hay dentro. Esto hace que veamos el menú en el que se nos indica que el juego está pausado y vemos el botón de continuar.

Este botón lo único que hace es quitar la pantalla de pause y poner el `Time.timeScale` a 1 para que el tiempo vuelva a pasar. Esto lo hacemos en la función `OnGUI` si detectamos que se a pulsado dicho botón de continuar, se ve como ponemos la variable de la que hemos hablado a 1 y modificamos la variable `l` para que no entremos mas a ese `if` ya que no nos interesa pues no queremos que el juego esté en el menú de pausa.

- Botón de controles: Este botón simplemente nos muestra un menú indicando como se hacen las cosas en el juego, movimiento disparo etc. Lo hacemos mediante el siguiente script asociado al botón de controles:

```
var l=0;
var mySkin2 : GUISkin;

function Update () {

    if(Input.touchCount > 0 ){
        for(var i : int = 0; i < Input.touchCount; i++){
            var touch : Touch = Input.GetTouch(i);
            if(touch.phase == TouchPhase.Began && guiTexture.HitTest(touch.position)){
                l=1;
                Time.timeScale=0;
            }
        }
    }
}

function OnGUI() {
    if(l==1) {
        Time.timeScale=0;
        GUI.skin=mySkin2;
        GUI.Box(Rect(0,0,Screen.width,Screen.height),"Mover tanque: Analogico");
        if (GUI.Button(Rect(1000, 600, 200, 80),"atras")) {
            l=0;
            Time.timeScale=1;
        }
    }
}
```



Como vemos es muy similar al script anterior, miramos si un dedo toca la pantalla, y si lo hace, miramos si toca el botón de controles. En ese caso vamos al menú en el que mostramos la ayuda para manejar el tanque y si le damos al botón atrás volvemos al juego para seguir donde nos habíamos quedado. Es el mismo menú que el de controles de la escena 1, sólo que igual al jugador se le ha olvidado hacer algo y lo quiere volver a mirar, es por eso que e vuelto a poner este botón para mirar de nuevo los controles.

- Botón de salir: Este botón simplemente nos retorna al menú inicial y lo he puesto porque puede que el jugador quiera empezar de nuevo o salir del juego si ya no quiere seguir jugando. El script que consigue esto es el siguiente:

```
static var ShowGUI = false;
function Start () {
ShowGUI = true;
}
function Update () {
    if(ShowGUI == true ){
        guiTexture.enabled = true;
    }else{
        guiTexture.enabled = false;
    }
    if(ShowGUI == true){
        if(Input.touchCount > 0 ){
            for(var i : int = 0; i < Input.touchCount; i++){
                var touch : Touch = Input.GetTouch(i);
                if(touch.phase == TouchPhase.Began && guiTexture.HitTest(touch.position)){
                    Application.LoadLevel(0);
                }
            }
        }
    }
}
```

De nuevo este script es muy similar a los anteriores pero cuando detecta que un dedo a tocado a este botón simplemente lo que hacemos es cargar la escena anterior, que es la del menú, de manera que volvemos automáticamente a dicha escena para salir del juego o volver a jugar.



- El siguiente botón es el de renacer: Este botón se usa para cuando te quedas atrancado o has ido por un camino equivocado o te has caído o cualquier otro caso que necesites volver al camino correcto. Lo que hacemos es poner una serie de checkpoints a lo largo del recorrido de manera que cuando pasamos con el tanque por alguno de ellos guardamos que ya hemos pasado por dicho checkpoint, por consiguiente cuando nos atasquemos o tengamos cualquier otro imprevisto y le demos a este botón apareceremos en este checkpoint. Si no hemos pasado por ninguno salimos donde hemos empezado si pulsamos el botón. Esto se hace para no tener que estar empezando cada vez que nos caemos o nos quedamos atascados como es lógico, sino sería difícil y acabaría resultando aburrido tener que empezar varias veces. El Script del botón es el siguiente:

```
function Update () {  
  
    var lol: Camera;  
    //The 'ShowGUI' enables you do either view or not view the actual GUI  
  
    if(Input.touchCount > 0 ){  
  
        for(var i : int = 0; i < Input.touchCount; i++){  
  
            var touch : Touch = Input.GetTouch(i);  
  
            if(touch.phase == TouchPhase.Began && guiTexture.HitTest(touch.position)){  
                //This space is where the action happens when you touch your custom GUI button on  
                //your device.  
                GameObject.Find("AAcanon1player").transform.position=GameObject.Find("AAcanon1player").GetComponent(cabina).posicion;  
                GameObject.Find("AAcanon1player").transform.eulerAngles=GameObject.Find("AAcanon1player").GetComponent(cabina).rotacion;  
            }  
        }  
    }  
}
```

Este código lo que hace es que si pulsamos el botón en cuestión, llevamos el tanque a la posición correspondiente al último checkpoint por el que hemos pasado. Los checkpoints simplemente son colliders en el mapa que detectan cuando un objeto ha pasado a través de él. Podemos ver uno de ellos en la siguiente imagen:



En esta imagen vemos el collider de uno de los checkpoints y como se puede apreciar el collider tiene marcada la opción Is Trigger, esto se hace para que detecte si algún objeto entra en el collider y mediante el script checkpoint invocamos a la función OnTriggerEnter que lo que hace es, al tener marcada la opción Is Trigger, detecta cuando algún objeto entra en su collider, de manera que si algo entra en este collider, miramos si lo que ha entrado es el tanque, y si lo es, establecemos que la posición de renacer sea la correspondiente a este checkpoint. Esto se ve en el script checkpoint asociado a ese objeto:

```
var respawn=0;
function OnTriggerEnter (myTrigger : Collider) {
    if(myTrigger.gameObject.name == "AAcanon1player"){
        switch(this.name)
        {
            case "checkpoint1":
                GameObject.Find("AAcanon1player").GetComponent.<cabina>().posicion=Vector3(-530.1075,28.18929,-1356.589);
                GameObject.Find("AAcanon1player").GetComponent.<cabina>().rotacion=Vector3(1.496777,181.1706,0.03057984);
                break;
            case "checkpoint2":
                GameObject.Find("AAcanon1player").GetComponent.<cabina>().posicion=Vector3(-530.1075,-26.2761,-1851.894);
                GameObject.Find("AAcanon1player").GetComponent.<cabina>().rotacion=Vector3(1.496777,181.1706,0.03057984);
                break;
            case "checkpoint3":
                GameObject.Find("AAcanon1player").GetComponent.<cabina>().posicion=Vector3(-463.9617,-114.501,-1476.96);
                GameObject.Find("AAcanon1player").GetComponent.<cabina>().rotacion=Vector3(1.496777,181.1706,0.03057984);
                break;
            case "checkpoint4":
                GameObject.Find("AAcanon1player").GetComponent.<cabina>().posicion=Vector3(-361.167,-37.30633,-2076.435);
                GameObject.Find("AAcanon1player").GetComponent.<cabina>().rotacion=Vector3(1.496777,181.1706,0.03057984);
                break;
        }
    }
}
```



Como vemos dependiendo del checkpoint que hayamos atravesado, la posición es una u otra.

- Botón disparo: Este botón es el encargado de realizar el disparo de la bala. El script asociado a este botón es el siguiente:

```
var progressBarEmpty : Texture2D;
var progressBarFull : Texture2D;
var mySkin : GUIStyle;
var barDisplay : float = 0;
var powerFactor=0.0;
var initialSpeed=10;
var maxPower=40.0;
var cont;
var projectile : Rigidbody;

function OnGUI()
{
    //msg = GUI.TextArea (Rect (10, 10, 200, 100), msg, 200);
    //if(GameObject.Find("Camera1").camera.enabled==true ) {
    GUI.BeginGroup (new Rect (300, 600, 400, 30));
    GUI.Box (Rect (0, 0, 400, 30),progressBarEmpty);

    GUI.BeginGroup (new Rect (0, 0, 400*barDisplay, 30));
    GUI.Box (Rect (0, 0, 400, 30),progressBarFull,mySkin);
    GUI.EndGroup ();

    GUI.EndGroup ();
}

function Update () {

if(Input.touchCount > 0 ){
for(var i : int = 0; i < Input.touchCount; i++){
var touch : Touch = Input.GetTouch(i);
if(guiTexture.HitTest(touch.position)){

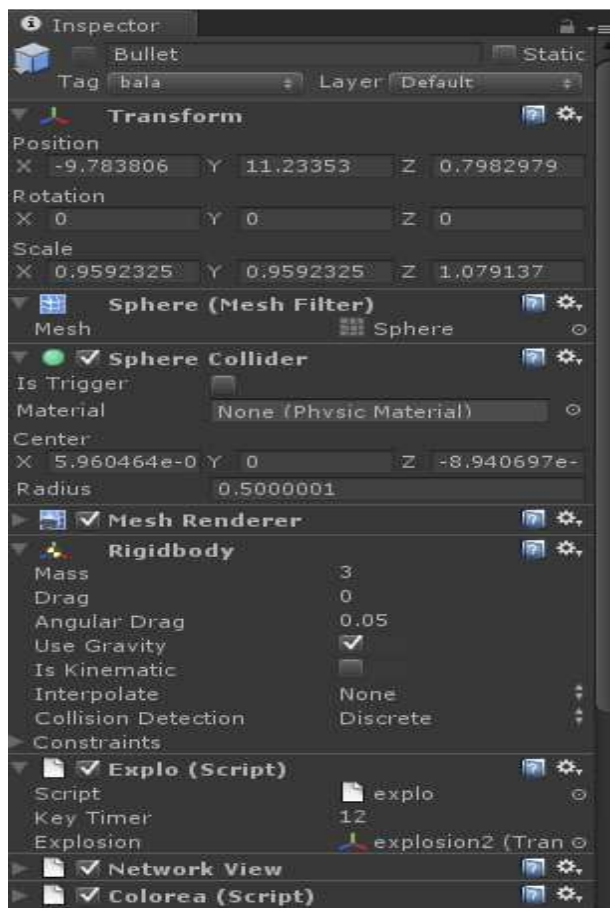
if(touch.phase == TouchPhase.Began || touch.phase == TouchPhase.Stationary || touch.phase == TouchPhase.Moved) {
powerFactor +=Time.deltaTime*initialSpeed;
barDisplay = powerFactor * 0.05;
cont=i;
}
else {
if(cont==1) {
powerFactor +=Time.deltaTime*initialSpeed;
barDisplay = powerFactor * 0.05;
var bala=GameObject.Find("canoncabina/canonmira/bala");
var instantiatedProjectile = Instantiate (projectile, bala.transform.position, bala.transform.rotation);
instantiatedProjectile.rigidbody.AddForce (-bala.transform.forward * maxPower * powerFactor *12 );
barDisplay=0;
powerFactor=0.0;
cont=0;
}
}
}
}
}
}
}
```



El script lo que hace es, en la función OnGUI dibujamos una barra que mostrará la potencia con la que vamos a lanzar la bala. Mientras dejemos el dedo pulsando el botón la barra se irá cargando pues estamos dando mayor potencia a la bala, y saldrá con una velocidad mayor cuanto más tiempo lo dejemos presionado. Si llega al tope cuando soltemos simplemente llevará el tope de velocidad.

En la función Update volvemos a mirar como en otros scripts si algún dedo toca este botón y si detectamos que ha sido así, mientras el Touch este en las fases de empezar mantener o movido lo que haremos es ir aumentando la barra de potencia de la bala y guardar en una variable cuanta velocidad le vamos a dar. Cuando soltemos el dedo lo que haremos será instanciar el proyectil Bullet que es nuestra bala, y darle la fuerza que hayamos guardado en la variable mencionada anteriormente.

El objeto Bullet es el que vemos en la imagen a continuación





Tiene unos scripts asociados que son los siguientes:

- El script Explo:

```
var KeyTimer : float = 5.0;
var explosion : Transform;

function Update () {

    //print(KeyTimer);
    if(KeyTimer>0) {
        KeyTimer -= Time.deltaTime;
        //print(KeyTimer);
    }
    else {
        explosion = Instantiate (explosion, transform.position, transform.rotation);
        Destroy(gameObject);
        Destroy(explosion, 3);
    }

}

function OnCollisionEnter(collision : Collision) {
    if(collision.gameObject.tag=="energy") {
        rigidbody.isKinematic=true; // stop physics
        transform.parent = collision.transform;
        //explosion = Instantiate (explosion, transform.position, transform.rotation);
        yield(WaitForSeconds(2));
        explosion = Instantiate (explosion, transform.position, transform.rotation);
        Destroy(gameObject);
        Destroy(explosion, 6);
    }
}
```

Este script lo que hace es que si el temporizador de la bala llega a 0 explota, y si antes de llegar a cero a detectado una colisión con uno de los objetos destruibles en la escena lo que hace es crearse una explosión y explotar también el objeto en cuestión.



- El otro script es Colorea:

```
var reddd:float = 1.0;
var luzzz:float = 1.0;
var KeyTimer : float = 3.0;
var KeyTimer1 : float = 3.0;
var KeyTimer2 : float = 3.0;
var KeyTimer3 : float = 3.0;

function start(){
    var KeyTimer : float = 3.0;
    var KeyTimer1 : float = 3.0;
    var KeyTimer2 : float = 3.0;
    var KeyTimer3 : float = 3.0;
}
function Update(){
    transform.renderer.material.SetColor("_Color", Color(reddd,greennn,blueee,luzz));
    transform.renderer.material.SetColor("_TintColor", Color(reddd,greennn,blueee,luzz));
    var luz1=Color(reddd,greennn,blueee,luzz);
    var luz2=Color(reddd,greennn,blueee,luzz);
    if(KeyTimer>0) {
        KeyTimer -= Time.deltaTime;
    }else{
    if(KeyTimer1>0) {
        KeyTimer1 -= Time.deltaTime;
        blueee -= blueee/20;
    }else{
    if(KeyTimer2>0) {
        KeyTimer2 -= Time.deltaTime;
        if (greennn >= 0.35){
            greennn -= greennn/20;
        }
    }else{
    if(KeyTimer3>0) {
        KeyTimer3 -= Time.deltaTime;
        greennn -= greennn/20;
    }else{
        reddd = 0;
        luzzz = 0;
        greennn = 0;
        blueee = 0;
    }
}
```

Este script simplemente lo que hace es ir cambiando la bala de color, empezando en blanco hasta al final llegar a ser rojo y explotar si no ha chocado con ningún objeto destructible.

A continuación muestro dos imágenes, la primera es la de la barra cargándose para lanzar la bala a una determinada potencia y la segunda es la bala ya lanzada:





Una vez lanzada la bala ponemos las variables de la potencia de la bala a cero y vaciamos la barra para que podamos volver a usarla y veamos todo correctamente de nuevo.

- Botón de la cámara: este botón lo que hace es ir cambiando entre las diferentes cámaras que posee el tanque, ya que se puede elegir entre varias cámaras según los gustos de cada jugador. Las diferentes cámaras que posee el tanque serán tratadas más tarde cuando hablemos sobre el tanque y sus scripts.



Este botón tiene asociado el script:

```
function Start () {  
  
    var cam = GameObject.Find("A\AcanonIplayer/cameras/Camera2").camera;  
    cam.enabled=true;  
}  
  
function Update () {  
  
    var lol: Camera;  
  
    if(Input.touchCount > 0 ){  
  
        for(var i : int = 0; i < Input.touchCount; i++){  
  
            var touch : Touch = Input.GetTouch(i);  
  
            if(touch.phase == TouchPhase.Began && guiTexture.HitTest(touch.position)){  
                if(GameObject.Find("A\AcanonIplayer/cameras/Camera1").camera.enabled== true) {  
                    GameObject.Find("A\AcanonIplayer/cameras/Camera1").camera.enabled=false;  
                    GameObject.Find("A\AcanonIplayer/cameras/Camera2").camera.enabled=true;  
                }  
  
                else if(GameObject.Find("A\AcanonIplayer/cameras/Camera2").camera.enabled== true) {  
                    GameObject.Find("A\AcanonIplayer/cameras/Camera2").camera.enabled=false;  
                    GameObject.Find("A\AcanonIplayer/cameras/Camera2k").camera.enabled=true;  
                }  
  
                else if(GameObject.Find("A\AcanonIplayer/cameras/Camera2k").camera.enabled== true) {  
                    GameObject.Find("A\AcanonIplayer/cameras/Camera2k").camera.enabled=false;  
                    GameObject.Find("A\AcanonIplayer/canoncabina/canonmira/Camera3").camera.enabled=true;  
                }  
  
                else if(GameObject.Find("A\AcanonIplayer/canoncabina/canonmira/Camera3").camera.enabled== true) {  
                    GameObject.Find("A\AcanonIplayer/canoncabina/canonmira/Camera3").camera.enabled=false;  
                    GameObject.Find("A\AcanonIplayer/canoncabina/canonmira/Camera4").camera.enabled=true;  
                }  
  
                else if(GameObject.Find("A\AcanonIplayer/canoncabina/canonmira/Camera4").camera.enabled== true) {  
                    GameObject.Find("A\AcanonIplayer/canoncabina/canonmira/Camera4").camera.enabled=false;  
                    GameObject.Find("A\AcanonIplayer/cameras/Camera5").camera.enabled=true;  
                }  
  
                else if(GameObject.Find("A\AcanonIplayer/cameras/Camera5").camera.enabled== true) {  
                    GameObject.Find("A\AcanonIplayer/cameras/Camera5").camera.enabled=false;  
                    GameObject.Find("A\AcanonIplayer/cameras/Camera1").camera.enabled=true;  
                }  
            }  
        }  
    }  
}
```

Este script detecta cuando un dedo ha pulsado este botón y en función de la cámara que estuviese activa, puesto que sólo se puede tener una activa a la vez, desactivamos esa cámara y activamos la siguiente. La última lo que hará será desactivarse y activar la primera y así todo el rato.

Ya hemos descrito los diferentes botones de la escena, ahora voy a describir el GUItexture que vemos en la parte de arriba de la pantalla:



Vemos que tenemos en la izquierda el tiempo restante para llegar a la meta, y a la derecha nuestra puntuación. El dibujo en el que están metidos estos dos números es un `GUITexture`:



Simplemente es aplicar el formato deseado a la `GUITexture` para que quede visualmente como deseamos e indicarle la posición en la que queremos que se encuentre.

Ahora comentaremos el número que vemos a la izquierda. Este número indica el tiempo que nos queda para poder llegar a la meta y funciona mediante el siguiente script:



```
private var textfield:GUIText;
// time variables
var mySkin2 : GUISkin;
var currentTime;|
var x=0;
var p=0;
var l=0;
function Update ()
{
    if(l==0) {
        if(x==0 ) {
            x=1;
            // retrieve the GUIText Component and set the text
            textfield = GetComponent(GUIText);
            UpdateTimerText();
            // start the timer ticking
            TimerTick();
        }
    }
}
function UpdateTimerText ()
{
    // update the textfield
    textfield.text = currentTime.ToString();
    if(currentTime<=0) {
        l=1;
    }
    // print(currentTime.ToString());
}

function TimerTick()
{
    // while there are seconds left
    while(currentTime > 0)
    {
        // wait for 1 second
        yield WaitForSeconds(1);
        // reduce the time
        currentTime--;
        UpdateTimerText();
    }
    // game over
}
function OnGUI() {
    if(l==1) {
        GUI.skin=mySkin2;
        GUI.Box(Rect(0,0,Screen.width,Screen.height),"GAME OVER\n\nNo has llegado a la meta a tiempo!");
        if (GUI.Button(Rect(800, 600, 350, 80),"Salir del juego")) {
            Time.timeScale=1;
            Application.Quit();
        }
        if (GUI.Button(Rect(300, 600, 350, 80),"Volver al menu")) {
            Time.timeScale=1;
            Application.LoadLevel(0);
        }
    }
}
}
```



Este script se encarga de hacer disminuir el tiempo restante cada segundo que pasa y mostrar el número actualizado en la parte de arriba a la izquierda. Simplemente consiste en ir esperando 1 segundo con `yield waitForseconds(1)` e ir restando 1 al tiempo restante.

Si el tiempo llega a ser cero, vemos en la función `OnGUI` que creamos una pantalla en la que diremos que el tiempo se ha agotado y mostraremos dos botones uno para volver al menú principal y otro para salir de la aplicación.

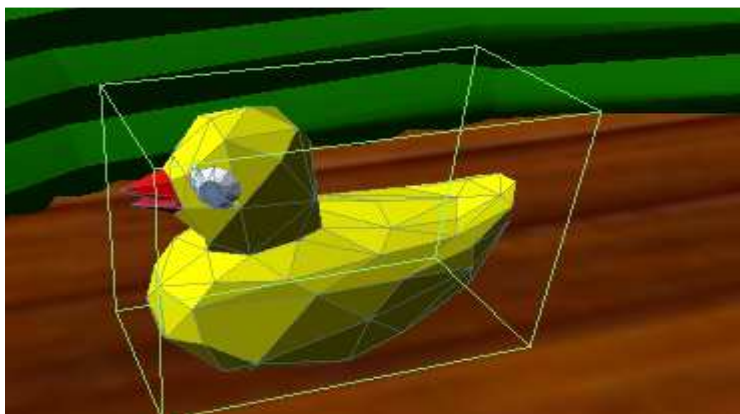
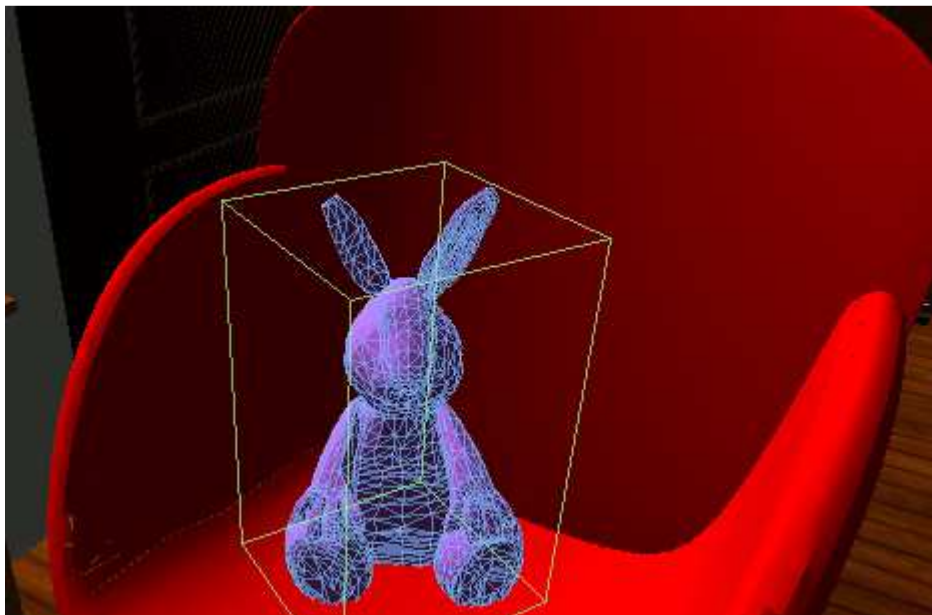
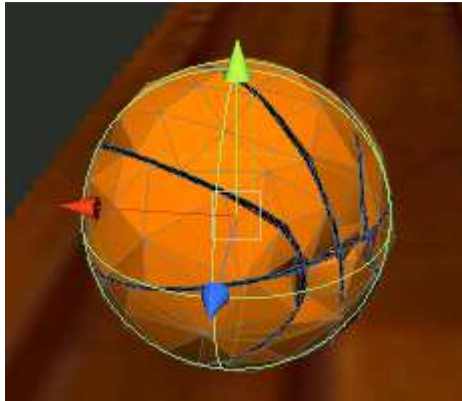
El otro número que vemos en la parte de arriba es el marcador



Y no lleva ningún script puesto que esta cifra se actualiza en los scripts que llevan los objetos destruibles en nuestra escena.

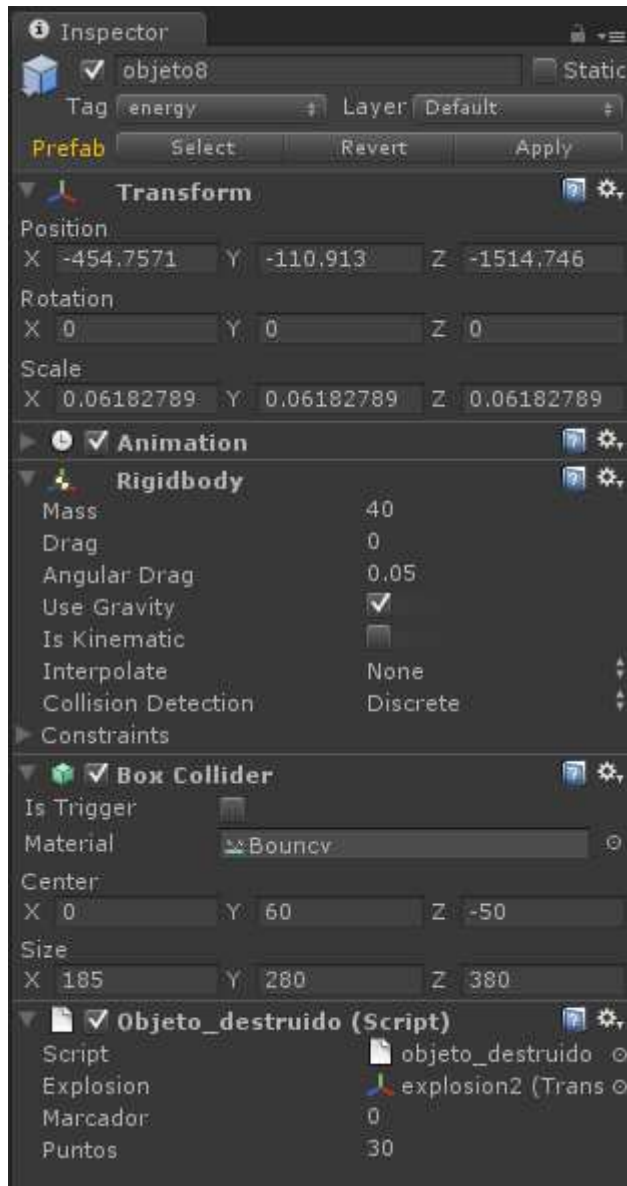


Estos objetos están presentes en la escena y son balones muñecos coches etc.





Y su inspector es de este estilo:



Como vemos en la imagen los objetos tienen asociados todos el mismo script y el código de este es el siguiente:



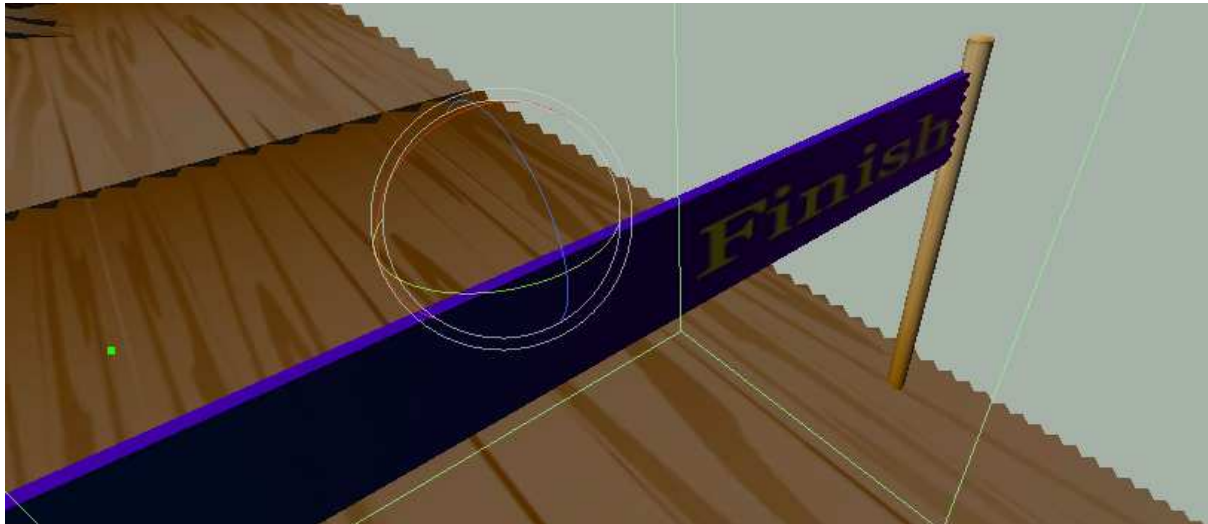
```
var explosion : Transform;
private var textfield:GUIText;
var marcador=0;
var puntos=30;
function OnCollisionEnter(collision : Collision) {
    if(collision.gameObject.tag=="bala") {
        yield(WaitForSeconds(2));
        textfield = GetComponent(GUIText);
        marcador=int.Parse(GameObject.Find("score(Clone)").guiText.text);
        switch(this.name)
        {
            case "objeto1":
                marcador=marcador+30;
                break;
            case "objeto2":
                marcador=marcador+50;
                break;
            case "objeto3":
                marcador=marcador+15;
                break;
            case "objeto4":
                marcador=marcador+20;
                break;
            case "objeto5":
                marcador=marcador+25;
                break;
            case "objeto6":
                marcador=marcador+30;
                break;
            case "objeto7":
                marcador=marcador+70;
                break;
            case "objeto8":
                marcador=marcador+60;
                break;
            case "objeto9":
                marcador=marcador+55;
                break;
            case "objeto11":
                marcador=marcador+90;
                break;
            case "objeto12":
                marcador=marcador+10;
                break;

            default:
                marcador=marcador+35;
        }
        GameObject.Find("score(Clone)").guiText.text = marcador.ToString();
        Destroy(gameObject);
    }
}
~
~
```



Este script simplemente lo que hace es que si una bala lanzada por el tanque ha impactado con el objeto, cuando se destruye actualiza la puntuación del marcador con el correspondiente valor asociado a cada objeto, ya que cada objeto tiene un valor diferente.

Ahora vamos a ver la meta, que se puede ver en el juego en la siguiente imagen:



Si el tanque llega aquí antes de que se le agote el tiempo, entonces ha cumplido el objetivo con éxito y el script que tiene asociado la meta nos muestra una pantalla con la puntuación y los botones para salir y volver al menú igual que cuando nos quedamos sin tiempo. El script de la meta es el siguiente:

```
var mySkin2 : GUISkin;
var l=0;
var puntuacion=0;
function OnTriggerEnter (other : Collider) {
    if(other.name.Equals("Aacanoniplayer")) {
        l=1;
        Time.timeScale=0;
    }
}

function OnGUI() {
    if(l==1) {
        GUI.skin=mySkin2;
        puntuacion=int.Parse(GameObject.Find("score(Clone)").guiText.text);
        GUI.Box(Rect(0,0,Screen.width,Screen.height),"Has llegado a la meta a tiempo!\n\nTu puntuacion es de: "+puntuacion);
        if (GUI.Button(Rect(800, 600, 350, 80),"Salir del juego")) {
            Time.timeScale=1;
            Application.Quit();
        }
        if (GUI.Button(Rect(300, 600, 350, 80),"Volver al menu")) {
            Time.timeScale=1;
            Application.LoadLevel(0);
        }
    }
}
```



Simplemente mira si lo que ha llegado a la meta es el tanque y en ese caso nos muestra la pantalla final que se muestra a continuación:

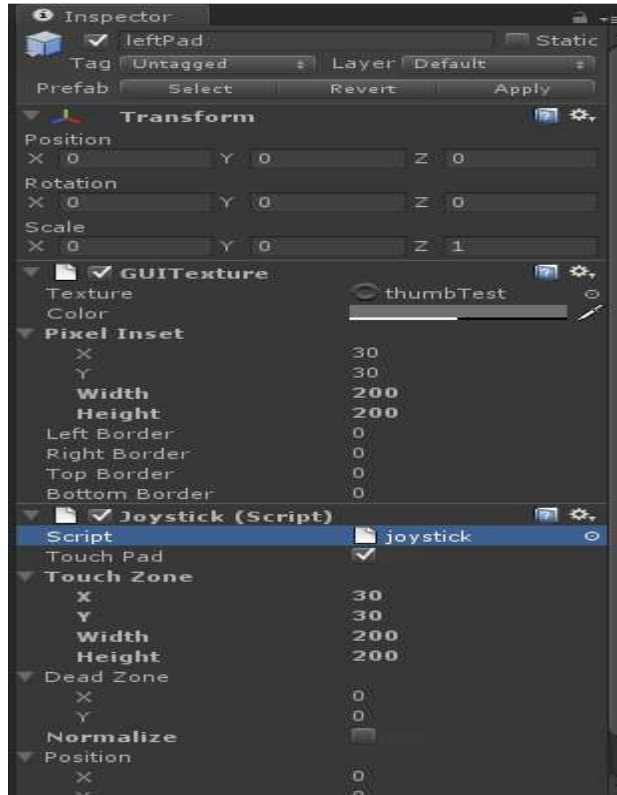


Ahora voy a comentar los sticks analógicos, son los dos círculos que salen abajo a la izquierda y abajo a la derecha respectivamente.





Se suelen ver de un color más transparente para no tapar parte del escenario y que molesten menos.



El script que tienen asociado estos Sticks es el mismo ya que sólo guardamos el valor de cuanto hemos desplazado el dedo una vez hemos pulsado dentro de uno de ellos. Lo que cambia es que el stick izquierdo esta relacionado con el movimiento del tanque y el derecho con la mira del tanque.

El script que llevan es el siguiente:



```
@script RequireComponent( GUITexture )

// A simple class for bounding how far the GUITexture will move
class Boundary
{
var min : Vector2 = Vector2.zero;
var max : Vector2 = Vector2.zero;
}

static private var joysticks : joystick[]; // A static collection of all joysticks
static private var enumeratedJoysticks : boolean = false;
static private var tapTimeDelta : float = 0.3; // Time allowed between taps

var touchPad : boolean; // Is this a TouchPad?
var touchZone : Rect;
var deadZone : Vector2 = Vector2.zero; // Control when position is output
var normalize : boolean = false; // Normalize output after the dead-zone?
public var position : Vector2; // [-1, 1] in x,y
var tapCount : int; // Current tap count

private var lastFingerId = -1; // Finger last used for this joystick
private var tapTimeWindow : float; // How much time there is left for a tap to occur
private var fingerDownPos : Vector2;
private var fingerDownTime : float;
private var firstDeltaTime : float = 0.5;

private var gui : GUITexture; // Joystick graphic
private var defaultRect : Rect; // Default position / extents of the joystick graphic
private var guiBoundary : Boundary = Boundary(); // Boundary for joystick graphic
private var guiTouchOffset : Vector2; // Offset to apply to touch input
private var guiCenter : Vector2; // Center of joystick
```

Estas son todas las variables necesarias para el funcionamiento de los joysticks. Ahora vamos a ver como tenemos que iniciarlos en la función Start



```
function Start()
{
// Cache this component at startup instead of looking up every frame
gui = GetComponent( GUITexture );

// Store the default rect for the gui, so we can snap back to it
defaultRect = gui.pixelInset;

defaultRect.x += transform.position.x * Screen.width;// + gui.pixelInset.x; // - Screen.width * 0.5;
defaultRect.y += transform.position.y * Screen.height;// - Screen.height * 0.5;

transform.position.x = 0.0;
transform.position.y = 0.0;

if ( touchPad )
{
// If a texture has been assigned, then use the rect ferom the gui as our touchZone
if ( gui.texture )
touchZone = defaultRect;
}
else
{
// This is an offset for touch input to match with the top left
// corner of the GUI
guiTouchOffset.x = defaultRect.width * 0.5;
guiTouchOffset.y = defaultRect.height * 0.5;

// Cache the center of the GUI, since it doesn't change
guiCenter.x = defaultRect.x + guiTouchOffset.x;
guiCenter.y = defaultRect.y + guiTouchOffset.y;

// Let's build the GUI boundary, so we can clamp joystick movement
guiBoundary.min.x = defaultRect.x - guiTouchOffset.x;
guiBoundary.max.x = defaultRect.x + guiTouchOffset.x;
guiBoundary.min.y = defaultRect.y - guiTouchOffset.y;
guiBoundary.max.y = defaultRect.y + guiTouchOffset.y;
}
}
```

En esta función guardamos la posición por defecto del GUI del joystick y les asignamos la zona de la textura como la zona en la que reconoceremos el dedo al tocar la pantalla.

Establecemos los límites para poder decidir si un dedo a tocado el joystick o no.



```
function Disable()
{
gameObject.active = false;
enumeratedJoysticks = false;
}

function ResetJoystick()
{
// Release the finger control and set the joystick back to the default position
gui.pixelInset = defaultRect;
lastFingerId = -1;
position = Vector2.zero;
fingerDownPos = Vector2.zero;

if ( touchPad )
gui.color.a = 0.025;
}

function IsFingerDown() : boolean
{
return (lastFingerId != -1);
}

function LatchedFinger( fingerId : int )
{
// If another joystick has latched this finger, then we must release it
if ( lastFingerId == fingerId )
ResetJoystick();
}
```

La función disable simplemente sirve para deshabilitar el joystick y la función ResetJoystick sirve para poner el joystick como estaba inicialmente y liberar las variables correspondientes a los dedos tocando el joystick.

Ahora procederemos a mostrar el código de la función update que esta implementado para que funcione tanto en el ordenador como en Android ya que las pruebas las iba haciendo para el ordenador y no tener que estar probando todo el rato en el móvil si me iba bien o mal.

Este código lo describo de manera general y simplemente va mirando si estamos en una plataforma táctil o no, y en función de eso mira si un dedo a tocado la pantalla o la ha tocado el ratón del ordenador, si esto ha ocurrido se modifica ligeramente el color del joystick y se calcula la x y la y de cuanto hemos arrastrado el dedo, estos valores van desde 1 hasta -1 en función de si movemos el dedo hacia arriba o hacia abajo, hacia la izquierda o hacia la derecha. Éstos datos son los que nos serán útiles después en los scripts del tanque para saber la velocidad a la que tiene que ir el tanque, la dirección o el movimiento de la mira según el joystick que estemos moviendo.



```
function Update()
{
if ( !enumeratedJoysticks )
{
// Collect all joysticks in the game, so we can relay finger latching messages
joysticks = FindObjectsOfType( joystick );
enumeratedJoysticks = true;
}

var count = 0;

if (Application.platform != RuntimePlatform.IPhonePlayer)
{
if(Input.GetMouseButton(0))
{
count = 1;
//Debug.Log("Mouse button: " + count);
}
}
else
{
count = Input.touchCount;
}

//var count = Input.touchCount;

// Adjustable
if ( tapTimeWindow > 0 )
tapTimeWindow -= Time.deltaTime;
else
tapCount = 0;

if ( count == 0 )
ResetJoystick(); ...
else
{
for(var i : int = 0;i < count; i++)
{
var touch : Touch;

var guiTouchPos : Vector2;
var fingerID : int;
var touchPosition : Vector2;

if (Application.platform != RuntimePlatform.IPhonePlayer)
{
guiTouchPos = Input.mousePosition - guiTouchOffset;
fingerID = 1;
touchPosition = Input.mousePosition;
}
else
{
touch = Input.GetTouch(i);

guiTouchPos = touch.position - guiTouchOffset;
fingerID = touch.fingerId;
touchPosition = touch.position;
}

var shouldLatchFinger = false;
if ( touchPad )
{
if ( touchZone.Contains( touchPosition ) )
shouldLatchFinger = true;
}
else if ( gui.HitTest( touchPosition ) )
{
shouldLatchFinger = true;
}

// Latch the finger if this is a new touch

```



```
if ( shouldLatchFinger && (lastFingerId == -1 || lastFingerId != fingerID))
{

if ( touchPad )
{
gui.color.a = 0.15;

lastFingerId = fingerID;
fingerDownPos = touchPosition;
fingerDownTime = Time.time;
}

lastFingerId = fingerID;

// Accumulate taps if it is within the time window
if ( tapTimeWindow > 0 )
tapCount++;
else
{
tapCount = 1;
tapTimeWindow = tapTimeDelta;
}

// Tell other joysticks we've latched this finger
for ( var j : joystick in joysticks )
{
if ( j != this )
j.LatchedFinger( fingerID );
}
}

if ( lastFingerId == fingerID )
{
// Override the tap count with what the iPhone SDK reports if it is greater
// This is a workaround, since the iPhone SDK does not currently track taps
// for multiple touches
if (Application.platform == RuntimePlatform.IPhonePlayer)
{
if ( touch.tapCount > tapCount )
tapCount = touch.tapCount;
}

if ( touchPad )
{
// For a touchpad, let's just set the position directly based on distance from initial touchdown
position.x = Mathf.Clamp( ( touchPosition.x - fingerDownPos.x ) / ( touchZone.width / 2 ), -1, 1 );
position.y = Mathf.Clamp( ( touchPosition.y - fingerDownPos.y ) / ( touchZone.height / 2 ), -1, 1 );
}
else
{
// Change the location of the joystick graphic to match where the touch is
gui.pixelInset.x = Mathf.Clamp( guiTouchPos.x, guiBoundary.min.x, guiBoundary.max.x );
gui.pixelInset.y = Mathf.Clamp( guiTouchPos.y, guiBoundary.min.y, guiBoundary.max.y );
}

if (Application.platform != RuntimePlatform.IPhonePlayer)
{
if (!Input.GetMouseButton(0))
{
ResetJoystick();
Debug.Log("Joystick Reset.");
}
}
else
{
if ( touch.phase == TouchPhase.Ended || touch.phase == TouchPhase.Canceled )
{
ResetJoystick();
Debug.Log("Joystick Reset.");
}
}
}
}
}
```



```
if ( !touchPad )
{
// Get a value between -1 and 1 based on the joystick graphic location
position.x = ( gui.pixelInset.x + guiTouchOffset.x - guiCenter.x ) / guiTouchOffset.x;
position.y = ( gui.pixelInset.y + guiTouchOffset.y - guiCenter.y ) / guiTouchOffset.y;
}

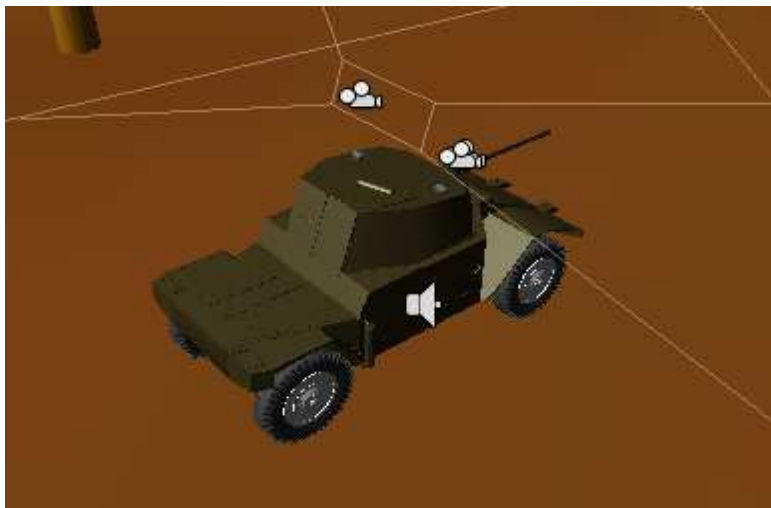
// Adjust for dead zone
var absoluteX = Mathf.Abs( position.x );
var absoluteY = Mathf.Abs( position.y );

if ( absoluteX < deadZone.x )
{
// Report the joystick as being at the center if it is within the dead zone
position.x = 0;
}
else if ( normalize )
{
// Rescale the output after taking the dead zone into account
position.x = Mathf.Sign( position.x ) * ( absoluteX - deadZone.x ) / ( 1 - deadZone.x );
}

if ( absoluteY < deadZone.y )
{
// Report the joystick as being at the center if it is within the dead zone
position.y = 0;
}
else if ( normalize )
{
// Rescale the output after taking the dead zone into account
position.y = Mathf.Sign( position.y ) * ( absoluteY - deadZone.y ) / ( 1 - deadZone.y );
}
}
```

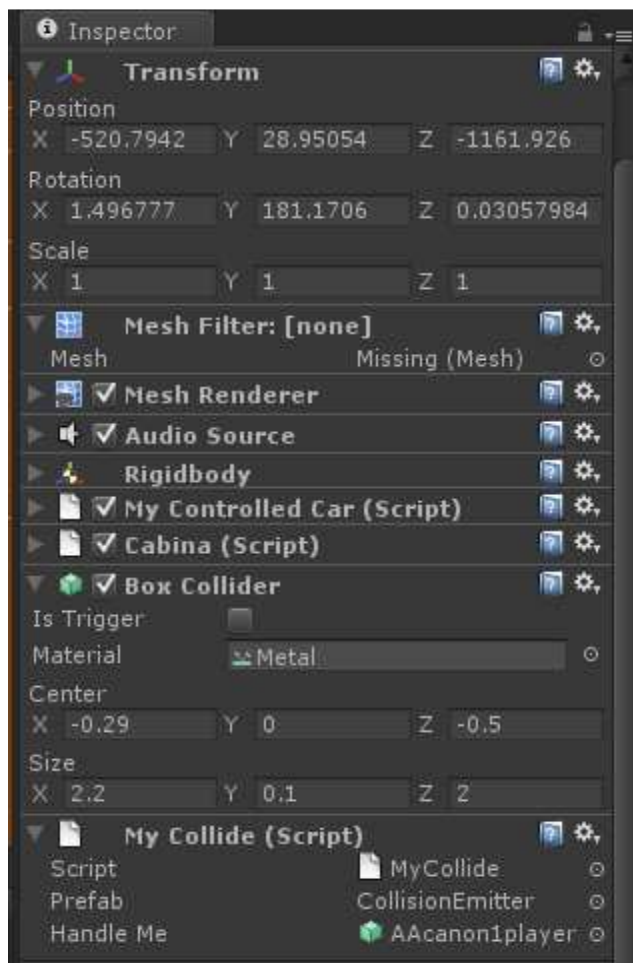
Este script es una adaptación de los joysticks del tutorial de penelope que fue uno de los que hice durante mi fase de aprendizaje de Unity 3d, y me serví de esos joysticks para adaptarlos a mi juego y modificar el script acorde a mis necesidades.

Por último ya sólo queda comentar el tanque

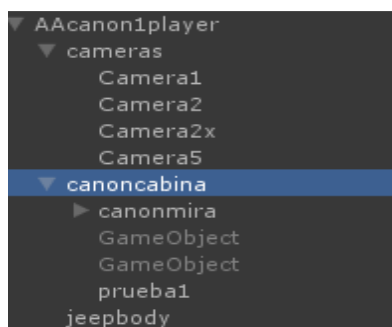




El tanque es un modelo en 3d que consta del cuerpo y de la cabina separados para poder mover la mira sin que se mueva el tanque.



Como vemos el tanque esta formado por un rigidbody y un collider y tiene asociados 3 scripts. Estos son los componentes del tanque, jeepbody es el modelo del tanque en 3d, cameras contiene todas las cámaras y todo esta dentro de aacanon1player que es el tanque en global.





Uno es el script Cabina que se encarga de aplicar la posición detectada por el joystick izquierdo y convertirla en el movimiento de la mira para poder apuntar a donde queramos disparar. Lo que hacemos es establecer unos topes para que el cañón no pueda atravesar al resto del tanque por eso tenemos una serie de sentencias if limitando los grados permitidos y lo único que hay que hacer es establecer el parámetro localEulerAngles en función del joystick izquierdo para que vaya girando en función de como movamos el joystick. Otra cosa que hacemos es que si nos caemos y volcamos volvemos al último checkpoint (anteriormente descritos).

```
function Start(){
    var gui = Instantiate(myGUITexture);
    var gui2 = Instantiate(myGUIText);
    var gui3 = Instantiate(myGUIText1);
}

function Update()
{
    r=GameObject.Find("AAcanonplayer/canoncabina").GetComponent(respawn).respawn;
    if(x==1) {
        Debug.Log("asdf");
        transform.position=posicion;
        transform.eulerAngles=rotacion;
        GameObject.Find("AAcanonplayer/canoncabina").GetComponent(respawn).respawn=0;
    }
    distance = Vector3.Distance(GameObject.Find("canoncabina/prueba1").transform.position, GameObject.Find("prueba2").transform.position);
    var distance2 = Vector3.Distance(GameObject.Find("canoncabina/prueba1").transform.position, GameObject.Find("prueba3").transform.position);

    var canonRotation = rotateJoystick.position;
    canonRotation.x *= rotateSpeed;
    canonRotation.y *= -rotateSpeed;
    canonRotation *= Time.deltaTime;
    var eulers2=GameObject.Find("canoncabina").transform.localEulerAngles;
    eulers2.y+=canonRotation.x*100;

    GameObject.Find("canoncabina").transform.localEulerAngles=eulers2;
    var eulers=GameObject.Find("canoncabina/canonmira").transform.localEulerAngles;
    eulers.x -=canonRotation.y * 100;
```




```
if((distance>=1.5) && (distance2>1.25)) {
    if(eulers.x >300) {
        if(eulers.x<=320) {
            eulers.x=320;
        }
    }
    else {
        if(eulers.x >40) {
            eulers.x=40;
        }
    }
}
else if ((distance<1.5) && (distance2>1.25)) {
    if(eulers.x >300) {
        if(eulers.x<=350) {
            eulers.x=350;
        }
    }
    else {
        if(eulers.x >40) {
            eulers.x=40;
        }
    }
}
else {
    if(eulers.x >300) {
        if(eulers.x<=340) {
            eulers.x=340;
        }
    }
    else {
        if(eulers.x >40) {
            eulers.x=40;
        }
    }
}
GameObject.Find("canoncabina/canonmira").transform.localEulerAngles=eulers;
```

Otro script que tenemos es el script MyCollide que lo usamos para crear el efecto de golpe contra una pared por ejemplo, hacemos un pequeño efecto de emisión de partículas simulando un choque contra una pared y saca un poco de humo como si hubiera habido un golpe.

El script que crea esto es el siguiente:

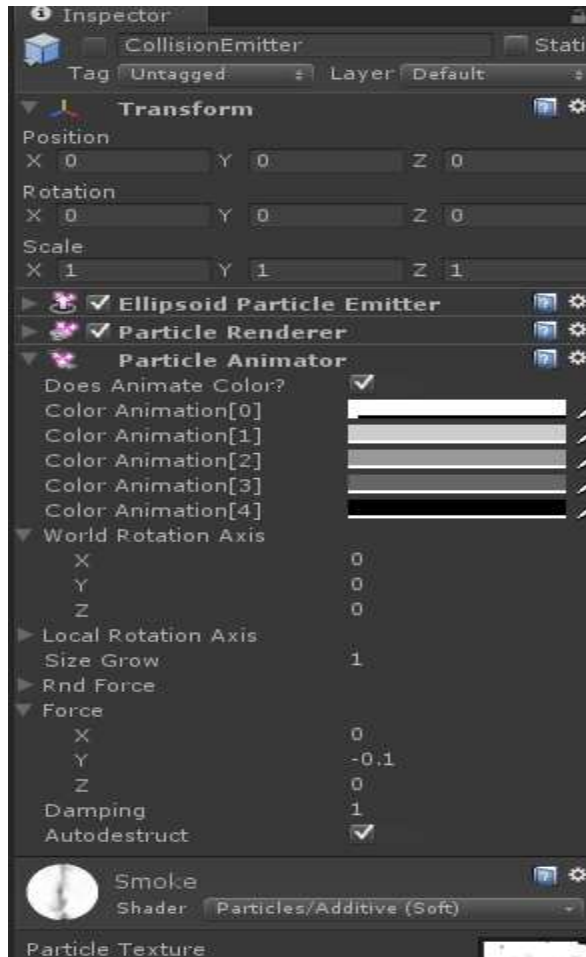
```
using UnityEngine;
using System.Collections;

public class MyCollide : MonoBehaviour {
    public GameObject prefab;
    public Collider handleMe;

    void OnCollisionEnter(Collision c) {
        int len = c.contacts.Length;
        foreach (ContactPoint p in c.contacts) {
            float mag = c.relativeVelocity.magnitude;
            if ((p.thisCollider == handleMe) || (p.otherCollider == handleMe)) {
                GameObject go = (GameObject) Instantiate(prefab, p.point, Quaternion.identity);
                ParticleEmitter e = go.particleEmitter;
                e.minEmission = mag * mag * 10 / len;
                e.maxEmission = mag * mag * 10 / len;
                e.emit = true;
                e.emitterVelocityScale = 0.01f * mag / len;
            }
        }
    }
}
```



Simplemente si algo entra en contacto con el tanque (esto se mira con la función OnCollisionEnter) realizamos el efecto de simulación de un golpeo contra una pared por ejemplo. El efecto de emisión de partículas es el siguiente



El último script que tiene asociado el tanque es MyControlledCar que lo que hace es a partir de los valores sacados del joystick derecho damos movimiento al tanque. Este movimiento es bastante complejo pues intentamos que vaya como si fuera con marchas empezando más despacio y poco a poco acelerando, y lo mismo para frenar. El movimiento se aplica a las 4 ruedas y vamos teniendo en cuenta la fuerza del motor para la aceleración y también para hacer los giros del tanque. Para acelerar y frenar usamos la posición del joystick derecho "y" y para girar la posición del joystick derecho "x".



```
using UnityEngine;
using System.Collections;

public class MyControlledCar : MyCar {

    // automatic, if true car shifts automatically up/down
    public bool automatic = true;
    public float shiftDownRPM = 1500.0f; // rpm script will shift gear down
    public float shiftUpRPM = 2700.0f; // rpm script will shift gear up
    public joystick moveJoystick;
    private Vector3 eulers3;

    void Update() {
        if (Input.GetKeyDown("page up")) {
            ShiftUp();
        }
        if (Input.GetKeyDown("page down")) {
            ShiftDown();
        }
        if (Input.GetKeyDown("g")) {
            automatic = !automatic;
        }
        if (Input.GetKeyDown("9")) {
            switch (wheelDrive) {
                case MyWheelType.Front : wheelDrive = MyWheelType.All; break;
                case MyWheelType.Back : wheelDrive = MyWheelType.Front; break;
                case MyWheelType.All : wheelDrive = MyWheelType.Back; break;
            }
            foreach (WheelData w in wheels) {
                WheelCollider col = w.col;
                col.motorTorque = 0f;
                col.brakeTorque = 0f;
            }
        }
    }

    void FixedUpdate () {
        float steer = 0; // steering -1.0 .. 1.0
        float accel = 0; // accelerating -1.0 .. 1.0
        bool brake = false; // braking (true is brake)

        if ((checkForActive == null) || checkForActive.active) {
            steer = moveJoystick.position.x;
            if((-moveJoystick.position.y)<0) {
                accel = -moveJoystick.position.y;
            }
            else {
                accel = -moveJoystick.position.y;
            }
            brake = Input.GetButton("Jump");
        }
        // handle automatic shifting
        if (automatic && (CurrentGear == 1) && (accel < 0.0f)) {
            ShiftDown(); // reverse
        }
        else if (automatic && (CurrentGear == 0) && (accel > 0.0f)) {
            ShiftUp(); // go from reverse to first gear
        }
        else if (automatic && (MotorRPM > shiftUpRPM) && (accel > 0.0f)) {
            ShiftUp(); // shift up
        }
        else if (automatic && (MotorRPM < shiftDownRPM) && (CurrentGear > 1)) {
            ShiftDown(); // shift down
        }
        if (automatic && (CurrentGear == 0)) {
            accel = - accel; // in automatic mode we need to hold arrow down for reverse
        }
        if (brake) {
            accel = -1f;
        }
        HandleMotor(steer, accel);
    }
}
```



Con esto hemos logrado un movimiento más o menos realista del tanque.

La parte de la cabina del tanque "canoncabina" posee un script para detectar si hemos volcado, en caso de volcar pone una variable a 1 y el script cabina del tanque en cada frame en la función update chequea si ese valor es 1. Si ese valor es 1 lo que hacemos es volver al último checkpoint por el que hayamos pasado. El script es el siguiente:

```
var respawn=0;
function OnTriggerEnter (myTrigger : Collider) {

    if(myTrigger.gameObject.name == "Cube_suelo"){
        respawn=1;
    }
    else {
        respawn=0;
    }
}
```

Por último voy a describir las diferentes cámaras para disfrutar del juego desde la vista que más nos guste.

- Cámara1:



- Cámara 2





- Cámara 2x



- Cámara 3



- Cámara 4



- Cámara 5





3.6.Problemas encontrados:

Al crear una aplicación sobre aplicaciones que en principio nos resultan desconocidas, como Unity 3d, sobre dispositivos que soporten Android suele traer consigo varios problemas que son los que ahora vamos a comentar.

Uno de los problemas principales fue la parte de la adaptación del de nuestra aplicación para que funcionara en Android. Al principio la aplicación fue creada para el ordenador usando ratón y teclado ya que era una plataforma conocida y era mucho más fácil para ir aprendiendo y probando la aplicación. Al pasarlo a Android había que cambiar muchas partes del código, en concreto, cada vez que haya que hacer un clic con el ratón o darle a una tecla del teclado, ya que teníamos que ponerlo en modo táctil con `input.touch` y demás clases ya tratadas anteriormente.

Otro problema fue que no dispuse de un móvil con Android hasta ya avanzada la aplicación, ya que antes poseía otro que tenía el sistema operativo Bada y no Android.

También hubo bastante problema con el tema del multitouch ya que los botones `GUIButtons` en Unity 3d no permiten al pasarlos a Android la opción, tuve que crear `GUITexturas` que simularan los botones y apañarlo un poco para que pudiera detectarlo.

Pero el mayor de los problemas sin duda fue el tema de que mi aplicación corriera de forma fluida sobre mi móvil con Android. Como es lógico un ordenador tiene mucha mas memoria Ram y mejor gráfica por tanto la tasa de frames alcanzaba los 200 o así. Sin embargo, en el móvil esto se reducía drásticamente y bajaba a una tasa inferior a 10, lo cual era inaceptable, ya que prácticamente se veía a cachos y funcionaba a tirones.

Así que estuve un tiempo investigando por foros y páginas web hasta que di con que el problema era el número de polígonos y vértices que estaban en la escena que renderizaba la cámara.

Por lo tanto me puse a intentar solucionarlo y la única solución que encontré a parte de la obvia que es quitar muchos objetos de la pantalla, era coger dichos objetos que formaban la habitación y en maya reducir los polígonos al máximo, con lo cual se empeora la calidad ya que cuantos menos polígonos más básico es el objeto pero manteniendo siempre un mínimo de calidad bajé los polígonos de los objetos todo lo que pude y al final conseguí hacer correr la aplicación sobre mi móvil de manera fluida y sin que funcionara a cachos. Esto puede no suceder en otros móviles, ya que si poseen peor hardware es posible que les vaya a una tasa de frames menor.



Conclusiones y líneas futuras:

4.1. Conclusiones técnicas:

Vamos a estudiar en este apartado si se han cumplido los objetivos del proyecto y si la planificación inicial de desarrollo de este ha sido alterada.

Estos puntos eran los objetivos del proyecto:

- Estudio de las herramientas, lenguajes y programas a utilizar en el diseño de la aplicación.
- Diseño de la aplicación.
- Implementación de la aplicación.
- Pruebas de la aplicación.
- Documentación de la aplicación para el correcto manejo del interface.

Para realizar el estudio de las herramientas, leí tutoriales básicos sobre el funcionamiento de Unity de la página <http://unity3d.com/gallery/demos/demo-projects> estuve mirando el del coche, el de shooter en tercera persona pero sobre todo me sirvió el de Penelope <http://u3d.as/content/unity-technologies/penelope-complete-project/1qZ>.

Este último trataba cosas como los joysticks etc. que realmente me resultaron muy útiles para la realización de mi proyecto. Gracias a estos tutoriales pude familiarizarme con Unity y hacer el proyecto en condiciones.

El tema del aprendizaje en C# y JavaScript no fue de una página específica puesto que ya sabía bastante de Java y ambos guardan bastantes similitudes con Java, así que las dudas y las cosas básicas de Unity las miraba en la pagina del Unity 3D o buscando en foros como www.unityspain.com/ por ejemplo.

Los programas externos que utilicé para poder crear la aplicación fueron Adobe Photoshop para tratar algunas texturas y Autodesk 3ds Max y Maya. El 3ds Max ya lo conocía de la asignatura "Gráficos y multimedia" de la Ingeniería técnica en Informática de Gestión de la Universidad Pública de Navarra (UPNA).

La implementación me costó bastante pues tuve que cambiar bastantes cosas entre JavaScript y C# y también la adaptación a Android era un poco complicada.

Las pruebas también fueron bastante costosas ya que cada vez que querías probar algo en el móvil tenías que recompilar la aplicación, pasarla al móvil, instala el .apk de tu aplicación y probarla. Si no funcionaba como esperabas tenías que volver a repetir el proceso y así sucesivamente.

No hemos pues un manual de usuario propiamente dicho en cuanto al tema de la documentación pero hemos ido describiendo lo que hacía cada cosa de cada escena y es bastante intuitivo, además que el usuario dispone de un botón de controles en todo momento en el cual se le explica como funciona de una manera clara y sencilla la aplicación.



4.2. Conclusiones personales:

El realizar este proyecto me ha dado una visión general de las opciones que tiene esta parte de mercado de la informática. El tema de los juegos se está extendiendo a una velocidad muy grande y gracias a programas como este, es posible realizar cosas bastante curiosas y subirlas al mercado de Android o a la appstore de una manera bastante sencilla y además se pueden hacer aplicaciones para Windows mac o Linux así como para XBOX o PS3.

Al hacer esta aplicación desde cero, he podido ver todas las dificultades que presenta realizar una aplicación de este tipo y como solucionar los problemas que se presentan gracias a los conocimientos adquiridos durante la carrera. Además he reforzado mis conocimientos sobre C# y JavaScript y puedo darles uso en futuras aplicaciones de cualquier estilo.



4.3. Líneas futuras:

Una vez concluido el proyecto vamos a explicar dos aspectos que deberían ser tratados en un futuro para mejorar la aplicación.

El tema del renderizado 3d de los objetos ya fue solucionado aunque me llevó bastante esfuerzo.

Lo que sí empecé a hacer pero por temas de tiempo no pude completar fue el tema de la opción multiplayer. Este tema si me gustaría hacerlo en un futuro ya que aporta mucha diversión el jugar contra otro usuario desde otro móvil y ver quien lo hace mejor. La dificultad que encontré fue el tema de ver lo que uno cambiaba en el otro móvil pero ya casi estaba solucionado así que en un futuro cercano me gustaría incluirlo ya que puede resultar muy divertido el jugar contra un amigo por ejemplo. También añadiría nuevas escenas con diferentes recorridos para aportar un poco más de dinamismo al juego.

Otra opción que se podría mejorar sería el movimiento del tanque, que se podría hacer aún mas realista pero es algo que resulta bastante costoso y creo que está bastante bien ahora mismo.

Por último otra característica a implementar en un futuro sería la opción de subirlo a Android Market pero hay aplicaciones gratuitas de mucha calidad y primero debería mejorarlo y hacer que funcionara mejor en móviles con pocas prestaciones de hardware antes de nada ya que sino no creo que tuviera mucho éxito.



Bibliografía:

En este apartado vamos a ver los libros, paginas web, etc que se han tenido como referencia para el aprendizaje y creación de la aplicación de nuestro proyecto.

Web oficial de Unity 3D:

- www.unity3d.com

Foro de consulta español sobre la aplicación Unity3D:

- www.unityspain.com

Tutoriales de Unity:

- <http://unity3d.com/gallery/demos/demo-projects>

Wiki inglesa sobre Unity 3D:

- <http://www.unifycommunity.com/>