

SOFTWARE LIBRE Y COMPROMISO SOCIAL

GRADO EN INGENIERIA INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CÓRDOBA



UNIVERSIDAD
DE
CÓRDOBA



SOFTWARE LIBRE EN EL DESARROLLO DE VIDEOJUEGOS

Autor:

Antonio Ruiz Roldán

30952977-z

i92ruroa@uco.es

Curso 2019/2020

Índice:

1.- Introducción.....	3
2.- Lista de software en videojuegos.....	4
1.a.- Unity.	4
1.b.- Construct 3.....	5
1.c.- GameMaker Studio 2.	5
1.d.- Godot Engine.	6
1.e.- Unreal Engine 4.....	6
1.f.- Stencyl.	7
1.g.- GDevelop.	7
1.h.- Defold.....	8
1.i.- Phaser 2d.	8
3.- La comunidad Unity.....	9
4.- Instalación de Unity en Mac.....	9
4.a.- Preparando el equipo.....	9
4.b.- Creación de Unity ID.	10
4.c.- Primeros pasos.....	11
5.- Características de Unity.	14
5.a.- Reutilización de código.	14
5.b.- Ocupación de componentes y creación de la interfaz de usuario.....	15
5.c.- Componentes Unity	16
5.d.- Otras características.....	18
6.- Manual de usuario básico.	19
6.a.- Juego realizado.	19
6.b.- Project Window.	20
6.c.- Hierarchy Window + Inspector Window	29
6.d.- Ejecutando el juego	33
7.- Extra. Otros juegos con Software Libre.	35
7.a.- Phaser.....	35
7.b.- Javascript + Html5.....	44
8.- Referencias.	54

1.- Introducción.

Los dos últimos años, he tenido la oportunidad de ser profesor en un Grado Superior de Formación Profesional, en el curso de Técnico Superior en Animaciones 3D, Juegos y Entornos Interactivos [1].

El área del que yo he sido responsable, ha sido la parte de entornos interactivos multimedia, y sobretodo, la programación de videojuegos.

En este sentido, el sector de los videojuegos es, seguramente, de los estudios multimedia más importantes de todos los existentes. Su cometido es el **diseño y desarrollo de videojuegos** para todo tipo de plataformas, ya sean videoconsolas, móviles u ordenadores personales. Se suele hacer el mismo producto para los diferentes dispositivos, aunque el terminal más utilizado son las videoconsolas (tanto portátiles como de salón). Y las aplicaciones para teléfonos móviles está creciendo con la popularización de los smartphones y las tablets.

La **evolución** de la industria de los videojuegos en las recientes décadas [2] ha sido exponencial, hasta el punto que hoy en día generan y mueven más dinero que la industria del cine y de la música juntas (llegó a generar más de 41200 millones de € la pasada década).

En el **desarrollo** de un videojuego participan una gran cantidad de profesionales de diferentes disciplinas, desde programadores, diseñadores, publicistas o gestores. Así, se puede decir que es la industria multimedia interactiva que genera más puestos de trabajo.

España se sitúa como el cuarto consumidor europeo de videojuegos (el primero es el Reino Unido), sin embargo, el número de productoras de videojuegos es escasa (sólo un 1% de la producción).

Actualmente, los últimos avances en materia de videojuegos se centran en la imagen 3D, en la detección del movimiento del usuario y en las aplicaciones para móviles.

Todo este conjunto de factores han hecho que elija este tema como **trabajo de la asignatura**.

He realizado la parte de un **videojuego 2D en Unity**. Y he desarrollado otros dos (en Phaser y Javascript) para demostrar que no hace falta conocimientos avanzados de programación para llevar a cabo un proyecto de estas características.

2.- Lista de software en videojuegos.

1.a.- Unity.

Se puede descargar en [Unity.com](https://unity.com) [3]. Desarrollado por Unity Technologies el 8 de Junio de 2005. Los autores principales fueron: David Helgason (CEO), Nicholas Francis (CCO), y Joachim Ante (CTO)

Utiliza Licencia de forma Gratuita y de Propietario. Yo soy profesor de Unity en un Grado Superior de Formación Profesional, y para mis alumnos ha sido necesario obtener una serie de licencia de Educación [4]

El lenguaje de programación en que se basa Unity es C#. Es una herramienta multiplataforma (tanto para Windows, Mac y todos los dispositivos móviles).

Para quienes no lo conozcan, Unity 3D es una herramienta que nos ayuda a desarrollar videojuegos para diversas plataformas mediante un editor y scripting para crear videojuegos con un acabado profesional. Esta herramienta está accesible al público en diferentes versiones, gratuita y profesional, cada cual con sus ventajas y limitaciones, evidentemente la más completa es la profesional pero es necesario hacer un desembolso que no todo el mundo puede permitirse y sobre todo si estamos comenzando a utilizar dicha herramienta.

1.b.- Construct 3.

Se puede descargar en <https://www.scirra.com/> [5] . Desarrollado por Scirra el 4 de Febrero de 2011.

El tipo de Licencia es de Propiedad o de código cerrado.

El lenguaje de programación en que se basa Construct es C++ y Javascript. Es una herramienta multiplataforma (tanto para Windows, Mac y todos los dispositivos móviles).

Esta es la mejor opción si nunca has escrito una línea de código en tu vida. Esta herramienta de desarrollo de juegos es completamente GUI-driven, lo que significa que todo es arrastrar y soltar. La lógica de juego y las variables se implementan utilizando las características de diseño proporcionadas por la propia aplicación. Lamentablemente, la codificación no está disponible incluso si deseas escribir código.

1.c.- GameMaker Studio 2.

Se puede descargar en <https://www.yoyogames.com/> [6]. Desarrollado por Mark Overmars el 15 de Noviembre de 1999. Actualmente es la empresa YoYo Games quien se ha hecho con sus derechos.

El tipo de Licencia es de Propiedad o de código cerrado.

El lenguaje de programación en que se basa GMS es Delphi y es un software que únicamente está disponible (en Inglés) para Microsoft Windows y opciones de exportación a Mac y otros Sistemas Operativos.

Soporta muchas características de calidad de vida interesantes, como la posibilidad de agregar compras en la aplicación a tu juego, análisis en tiempo real sobre cómo los usuarios juegan tu juego, fuente Control, redes multijugador y extensibilidad a través de extensiones de terceros. También incorpora editores para imágenes, animaciones y sombreadores.

1.d.- Godot Engine.

Se puede descargar en <https://godotengine.org/> [7]. Desarrollado inicialmente por la empresa OKAM Studios en el año 2001. Actualmente es la propia comunidad quien lo está desarrollando.

Hace uso de Licencia MIT o X11 (originada en el Instituto Tecnológico de Massachusetts).

El lenguaje de programación en que se basa C y C++. Puede desplegarse en múltiples plataformas directamente, incluyendo Windows, Mac, Linux, Android, iOS y HTML5. No se necesitan compras o licencias adicionales, aunque pueden aplicarse algunas restricciones (como la necesidad de estar en un sistema Mac para desplegar un binario Mac).

Godot itera sorprendentemente rápido para un motor de juego. Hay al menos un lanzamiento importante cada año, lo que explica cómo ya tiene tantas características excelentes: física, post-procesamiento, redes, todo tipo de editores integrados, depuración en vivo y recarga en caliente, control de código fuente y más.

1.e.- Unreal Engine 4.

Se puede descargar en <https://www.unrealengine.com/en-US/> [8] Desarrollado por la compañía Epic Games en 1998.

Hace uso de Licencia Privativa

El lenguaje de programación es C++. Y La versión actual está diseñada para las plataformas Microsoft Windows, macOS, Linux, SteamOS, HTML5, iOS, Android, PlayStation 4, Nintendo Switch, Xbox One SteamVR/HTC Vive, Oculus Rift, PlayStation VR, Google Daydream, OSVR y Samsung Gear VR.

De todos los software aquí nombrados, este es el más profesional. Fue creado a partir de cero por los genios detrás de la franquicia de Unreal – la gente que sabe lo que se necesita en un motor de plataforma superior y lo que se necesita para ofrecer características de próxima generación. Baste decir que saben exactamente lo que están haciendo. El Canal YouTube de UE4 tiene más de 800 videos

que te llevan a través de cada pulgada del motor, y la mayoría de esos videos tienen entre 20 y 60 minutos de duración. Eso es más contenido de lo que obtendría de un curso de un semestre en la universidad. Si necesita una guía paso a paso, UE4 lo ha cubierto.

1.f.- Stencyl.

Se puede descargar en <http://www.stencyl.com/> [9] Desarrollado por Jonathan Chung en 2011 (llamado originalmente StencylWorks)

Hace uso de Licencia Privativa

Programado en Java, ActionScript, Objective-C, C++ y Haxe. Y permite crear videojuegos con gráficos 2D para ordenadores, dispositivos móviles, y páginas web.

describen el proyecto, su funcionalidad y su utilidad. Stencyl no es el típico software de creación de juegos; es un conjunto de herramientas intuitivo que acelera el flujo de trabajo y luego se quita del camino. Nos ocupamos de lo esencial, para que pueda concentrarse en lo que es importante: hacer que su juego sea suyo. Los mejores juegos de Stencyl han llegado a los primeros puestos en la App Store y Google Play, mientras que aparecen en la sección "Mejor juego nuevo" en sus respectivas tiendas.

1.g.- GDevelop.

Se puede descargar en <https://gdevelop-app.com/> [10].
Desarrollado en 2008 por Florian Riva

Hace uso de Licencia MIT o X11 (originada en el Instituto Tecnológico de Massachusetts).

GDevelop está desarrollado en C++ y Javascript, y se ejecuta en distribuciones Windows, macOS y las más recientes de Linux. También puede probarlo en línea usando Chrome, Firefox u otro navegador web o móvil reciente.

Es un software que permite crear todo tipo de juegos en 2D sin usar un lenguaje de programación que quiera ser accesible para los más pequeños gracias a un sistema de condiciones y acción en forma

de bloques llamados eventos. El programa es completamente gratuito y los juegos creados con Windows, GNU / Linux o en la Web (juegos en HTML5) y pertenecen completamente al usuario que tiene el derecho de hacer un uso comercial sin pagar regalías o licencia.

1.h.- Defold.

Se puede descargar en <https://defold.com/> [11]. Comenzó con Ragnar y Christian hablando sobre la creación de juegos en Avalanche, la compañía de juegos AAA donde ambos trabajaban, en 2010.

Utiliza Licencia de forma Gratuita y de Propietario.

Está desarrollado en Lua, y se puede ejecutar en iOS, Android, HTML5, Windows, OSX, Linux.

Defold está hecho para ser una plataforma de producción de juegos profesional para ayudar a los equipos de juego a diseñar, construir y enviar juegos. No es una solución que lo abarque todo para todo. No hay componentes complejos listos para usar disponibles. En cambio, creemos que el trabajo de Defold es capacitar a los equipos de juego con herramientas colaborativas simples y poderosas. Esto significa que a menudo tiene que hacer un poco más de trabajo usted mismo, pero también significa que el camino hacia su objetivo es más claro.

1.i.- Phaser 2d.

Se puede descargar en <https://phaser.io/> [12]. Fue creado por Richard Davey en Abril de 2013. Y posteriormente la empresa Photo Storm se hizo cargo del Software.

Hace uso de la Licencia MIT o X11 (originada en el Instituto Tecnológico de Massachusettss).

Phaser es un Framework de Javascript que utiliza también Typescript y Html5 para su funcionamiento. Se puede visualizar en cualquier tipo de navegador o dispositivo.

Phaser es un framework de juegos de escritorio y Mobile HTML5 de código abierto principalmente. Incluye un conjunto robusto de documentación, características y ejemplos para que pueda avanzar rápidamente hacia un juego de trabajo. Es compatible con WebGL, a través del motor de renderizado Pixi.js , e incluye un respaldo Canvas para soporte en dispositivos más antiguos.

3.- La comunidad Unity.

Unity cuenta actualmente con una de las comunidades, dentro del mundo de los Video Juegos, más grande en el mundo.

A nivel Nacional, podemos encontrar **Unity Spain** [13] en la que aparte de Tutoriales y Recursos, podemos encontrar muchos foros dónde encontrar solución a cualquier tipo de problema o incluso eventos en los que se encuentran diseñadores y desarrolladores de videojuegos.

Aparte, hay una amplia oferta y demanda de puestos de trabajo para este tipo de sector.

Como es obvio, existe la posibilidad de las donaciones por parte de los usuarios, para que la comunidad pueda existir.

También, en la web oficial de **Unity** [14] podemos encontrar muchos de estos recursos: Foros, ayuda en vivo, Respuestas a problemas, etc.

4.- Instalación de Unity en Mac.

4.a.- Preparando el equipo.

Como hemos comentado, Unity se puede descargar para Windows y para Mac.

En nuestro caso, vamos a trabajar sobre este segundo sistema operativo.

Unity requiere de ciertas características y potencia del ordenador para poder trabajar. Sin embargo, en el ordenador en el que hemos desarrollado el videojuego (un ordenador de 2012) ha funcionado sin ningún tipo de problema.

Hemos trabajado sobre macOS High Sierra (versión 10.13.6).

Procesador 2,9 GHz Intel Core i5

Memoria 8 GB 1600 MHZ DDR3

Hay que decir, que la versión actual de Unity es la 2019.3.11.

Sin embargo, para este proyecto hemos usado la 2017.3.0, la cual podremos descargar desde el siguiente enlace o escribiendo en Google "unity descarga versiones" y pinchando en el primer enlace.

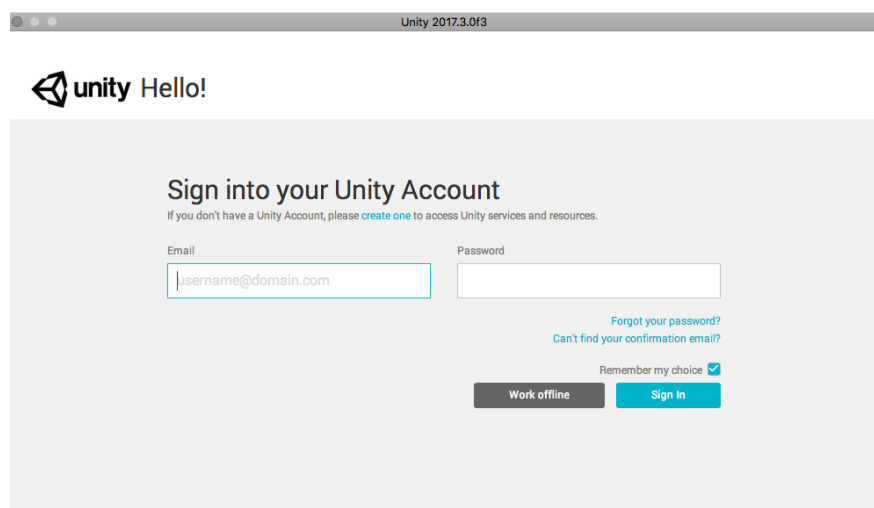
<https://unity3d.com/es/get-unity/download/archive> [15]

Bajaremos el archivo y seguiremos los pasos de instalación, que son muy sencillos.

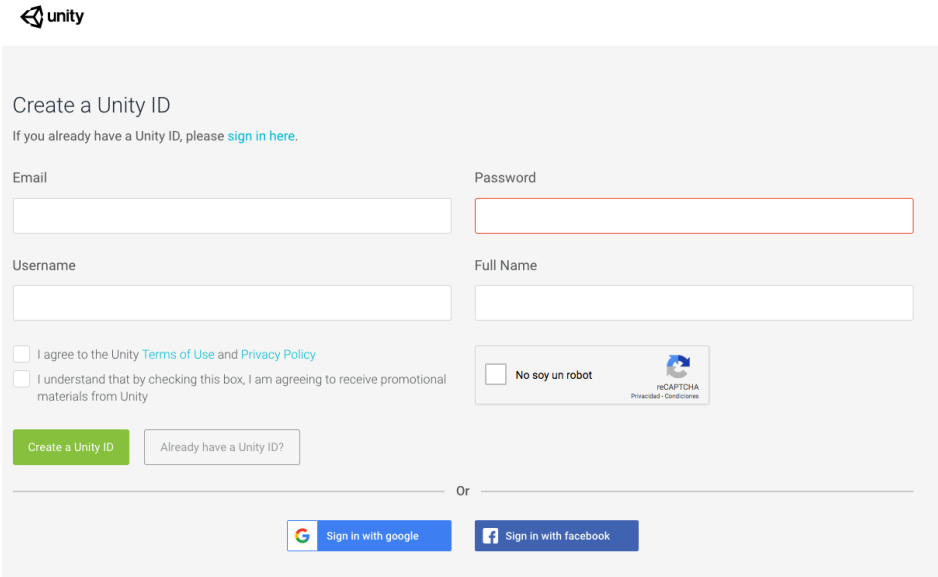
4.b.- Creación de Unity ID.

Para poder programar en Unity, es necesaria una cuenta en Unity para poder comenzar a trabajar (se obtiene un ID de desarrollador)

Al iniciar el programa por primera vez, la primera pantalla que aparece es la de Loguearse con la cuenta de Unity o crear una nueva.



Si lo que queremos es crear una nueva cuenta, al pinchar se abre el formulario de registro en nuestro navegador [16]. Se puede registrar por e-mail, por cuenta de Google o de Facebook.

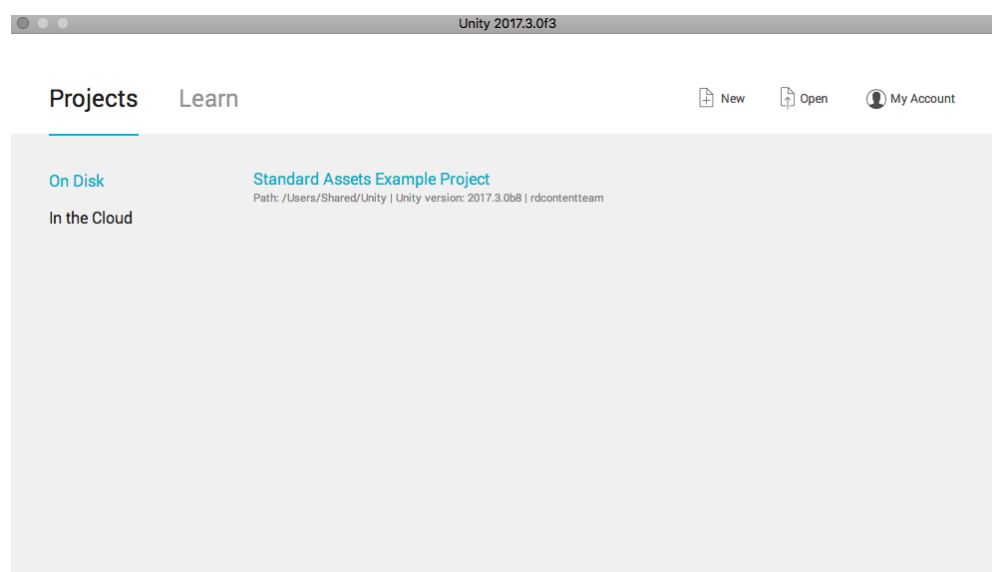


The image shows the 'Create a Unity ID' registration form. At the top left is the Unity logo. Below it, the title 'Create a Unity ID' is followed by a link: 'If you already have a Unity ID, please [sign in here](#).' The form contains four input fields: 'Email', 'Password', 'Username', and 'Full Name'. Below the 'Email' field are two checkboxes: 'I agree to the Unity [Terms of Use](#) and [Privacy Policy](#)' and 'I understand that by checking this box, I am agreeing to receive promotional materials from Unity'. To the right of these is a reCAPTCHA widget with the text 'No soy un robot'. At the bottom left is a green 'Create a Unity ID' button, and next to it is a link 'Already have a Unity ID?'. Below these is a horizontal line with 'Or' in the center. At the bottom are two buttons: 'Sign in with google' and 'Sign in with facebook'.

Una vez creada la cuenta, ya se puede introducir las credenciales en nuestro programa.

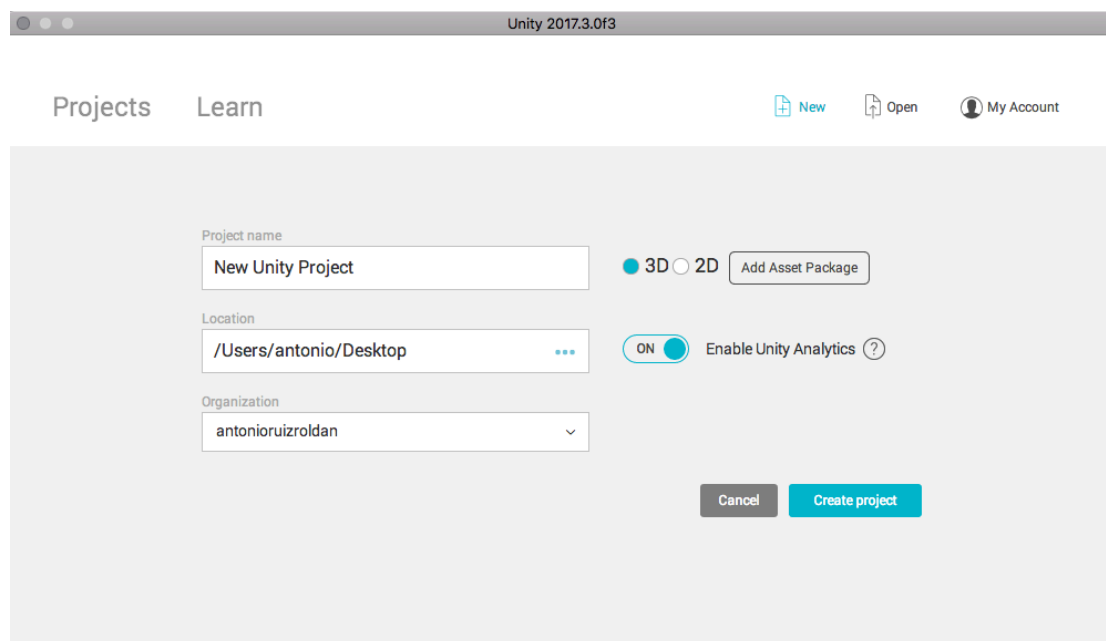
4.c.- Primeros pasos.

Una vez abierto nuestro programa e introducido nuestro e-mail y contraseña, aparece la pantalla principal del programa.



Aparecerá una interfaz donde podremos ver los proyectos creados, y en otra pestaña tutoriales que pueden ser de interés para el desarrollador.

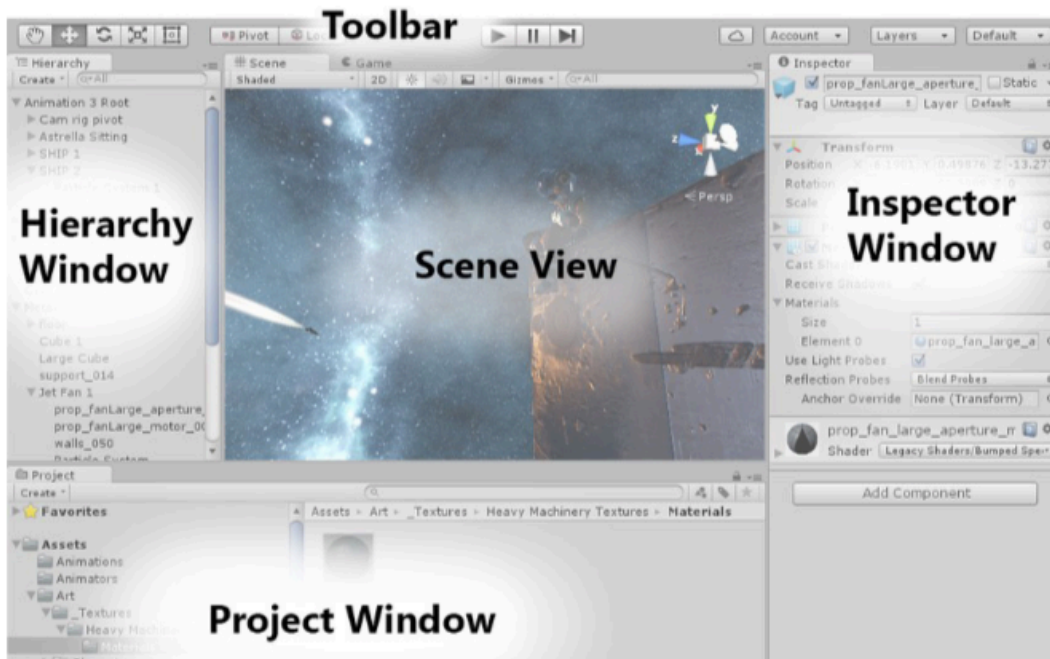
En la esquina superior derecha podremos observar tres botones: New, Open y MyAccount; pulsaremos en New para empezar un nuevo proyecto.



Añadiremos un nombre y una ruta de guardado y clicaremos en Create Project (no hay que preocuparse ahora por si es un proyecto 3D o 2D, ya que podremos cambiarlo más adelante).

También se pueden activar una serie de analíticas de Unity o añadir desde el principio paquetes y funcionalidades. Nosotros no vamos a hacerlo en este caso.

Como se puede ver en la siguiente imagen, la pantalla principal se puede dividir en 5 áreas bien diferenciadas: Toolbar, Hierarchy Window, Scene View, Inspector Window y Project Window.



En el **Toolbar** (o barra de herramientas) se encuentran aquellas herramientas que nos permiten trabajar con Unity. Las más importantes son:

- La mano, para movernos por el entorno.
- La herramienta de mover, para mover los elementos.
- La herramienta de rotación.
- La herramienta de escalado.
- O la herramienta Rect Tool, para dar forma a nuestra interfaz de usuario (que veremos en el siguiente punto).
- Botón de Play, Pause y Step, para iniciar nuestra escena.
- Y en la parte derecha, como hemos comentado, se puede cambiar el diseño (Layout) de nuestro programa. Ahora mismo está en Default, pero se puede cambiar según las necesidades.

Project Window (ventana del proyecto): en esta sección tendremos que añadir todos los assets (recursos) que queramos utilizar en nuestro proyecto, ya sea un personaje 2D, una textura o una animación. Si se da al botón derecho, se puede ver que se pueden crear carpetas, Scripts, Borrar, etc.

Hierarchy Window (ventana de jerarquía): en esta ventana se encuentran todos los elementos que están dentro de nuestra escena, es decir, todo lo que estamos usando en el nivel del juego que tengamos abierto. Si por

ejemplo alguna de las imágenes que está en la ventana del proyecto no se mueve en la ventana de jerarquía, no aparecerá en nuestro juego.

Scene View (vista de escena): nos permite observar a nivel visual la aplicación o nivel que estemos creando, ya sea la interfaz de una aplicación móvil o la construcción de niveles de un videojuego.

Inspector Window (ventana del inspector): nos muestra todas las opciones que tiene el elemento que tengamos seleccionado en la jerarquía (a lo que llamaremos GameObject)

5.- Características de Unity.

5.a.- Reutilización de código.

Una de las principales características de Unity es que tiene diferentes propiedades para la reutilización de código, para que a la hora de programar ya se pueda aprovechar muchos de estos elementos.

A continuación vamos a describir cada una de ellas:

Librerías de funciones

Las librerías de funciones son compilaciones de implementaciones funcionales y están codificadas en un lenguaje de programación. Están preparadas para ser utilizadas por otras aplicaciones de forma simultánea. Además, se pueden complementar y vincular a otros programas.

- **Bibliotecas estáticas:** son ficheros que contienen archivos de código objeto empaquetados que pueden ser relocalizados junto con el resto de los ficheros de código objeto. Es decir, las bibliotecas estáticas tienen como función principal contener los ficheros de código objeto que no se diferencian de los demás.
- **Bibliotecas dinámicas:** son ficheros que contienen un código objeto construido sin importar su ubicación.
- **Bibliotecas remotas:** contienen y usan ejecutables separados y, a través de llamadas de procedimiento remoto, los llaman sobre la red a otra computadora.

Componentes de Software

Los componentes de un software son las partes de un programa, es decir, los módulos que este contiene. Cada módulo tiene una o más tareas definidas que debe realizar dentro del programa y están organizados por niveles. Esta ordenación es jerárquica, por lo que uno de los módulos se encargará de llamar a esos otros que se encuentran en niveles inferiores, y así en orden de jerarquía.

Por otro lado, los módulos son pequeños e independientes. Esto facilita el trabajo, ya que no será necesario conocer los demás módulos a la hora de diseñar uno.

Comportamiento

Cuando hablamos de comportamiento de un objeto, entendemos que el comportamiento está vinculado a su funcionalidad (determinada por la responsabilidad) y delimita las operaciones que puede realizar o las que puede responder ante mensajes enviados por otros objetos.

En POO se puede reutilizar el código porque la responsabilidad de los objetos está definida y es concreta. Por eso es tan importante definir y delimitar el comportamiento de los objetos de las clases involucradas en el diseño de aplicaciones (como hemos hecho anteriormente, al añadirles un nombre cualquiera a nuestras variables para poder reutilizarlas después).

5.b.- Ocupación de componentes y creación de la interfaz de usuario

Otra de las características fundamentales de Unity, es que es un Software Multidispositivo. A la hora de exportar los juegos, se puede hacer que se ejecute en Windows, Mac, dispositivos móviles, incluso en consolas.

Para ello, se tiene en cuenta el concepto de **Interfaz Gráfica de Usuario [17]**. Que es un programa informático que utiliza imágenes y objetos gráficos para representar las acciones y la información de la interfaz, es decir, es un entorno visual que facilita la comunicación entre el ordenador y el sistema operativo.

Hay diferentes tipos de interfaces gráficas y su elección dependerá del objetivo que vayan a cumplir.

- **Graphical User Interfaces (GUI):** son las habituales en videojuegos y en realidad virtual.
- **Zooming User Interface (ZUI):** se usan en investigación y mezclan 2D y 3D.
- **Interfaz de usuario de pantalla táctil:** son GUI de uso específico y se manejan mediante una pantalla táctil. En los últimos años se han implementado en múltiples sectores, desde cajeros automáticos hasta tiendas de autoservicio.
- **Natural User Interface (NUI):** se interactúa con el sistema sin dispositivos de entrada, es decir, se usan los dedos (táctil) en vez del ratón o el teclado.

El sistema UI permite crear interfaces de usuario rápidas e intuitivas. A continuación, se introducen las características principales del sistema UI de Unity:

- **Rect Tool:** todos los elementos de UI se representan a través de un rectángulo que permite que el usuario pueda manipularlo en la vista de la escena a través de Rect Tool. Esta opción se encuentra en la barra de herramientas. Rect Tool se utiliza para mover, cambiar el tamaño y rotar los elementos UI. Utiliza el modo de pivote actual y espacio, y se utiliza tanto para objetos 2D como UI y 3D.
- **Rect Transform:** además de las funcionalidades generales de un componente Transform general (mover y escalar), también añade el ancho y la altura para especificar las dimensiones del rectángulo.
- **Pivot:** permite que el pivote se pueda mover en la vista de la escena a través de la configuración a modo Pivote.

5.c.- Componentes Unity

Tenemos que destacar la gran cantidad de componentes que tiene Unity para darle forma a nuestro juego. Componentes, la gran mayoría, que se basan en propiedades de la Física (como la gravedad, colisiones, etc).

Estos componentes los utiliza los **GameObjects** de Unity [18]. Que son objetos fundamentales que representan personajes, propiedades del juego, y el escenario. Estos no logran nada por sí mismos pero funcionan

como contenedoras para estos componentes, que implementan la verdadera funcionalidad.

El componente que siempre se añade al crear un GameObjects nuevo es el **Transform Component** (transform), que podremos ver en la sección del inspector al seleccionar el GameObject.

Tiene múltiples funcionalidades, tales como definir la posición, la rotación y la escala del GameObject en el juego. Cada GameObject tiene un Transform que determina la posición en coordenadas X, Y y Z; la rotación alrededor de los ejes X, Y y Z en grados; y la escala a lo largo de los ejes X, Y y Z.

Además del Transform, cada GameObject puede tener otros componentes como, por ejemplo:

- **Camera Component:** permite que el jugador vea el mundo creado. Las cámaras se pueden manipular y personalizar para crear un juego único.
- **GUI Layer:** se adjunta a la cámara para habilitar la renderización de GUIs 2D. Renderiza todas las GUI Textures y GUI Texts de la escena.
- **Audio Listener:** actúa como un micro, recibiendo input de cualquier Audi Source en la escena, y reproduce el sonido a través de los altavoces del computador. Se debe tener en cuenta que solo funcionará si se agrega.
- **Box Collider:** se utiliza para crear una colisión en el objeto, es decir, para que al acercarnos colisionemos contra él y nos impida atravesarlo, como si de un fantasma se tratara.
- **Rigidbody:** le da física al GameObject haciendo, por ejemplo, que le afecte la gravedad.
- Etc...

Aparte de todos estos, nosotros podemos programar nuestros propios componentes y funcionalidades. Para ello, Unity hace uso de los lenguajes de programación C# y TypeScript. No hay que tener grandes conocimientos de programación para programar un videojuego.

5.d.- Otras características

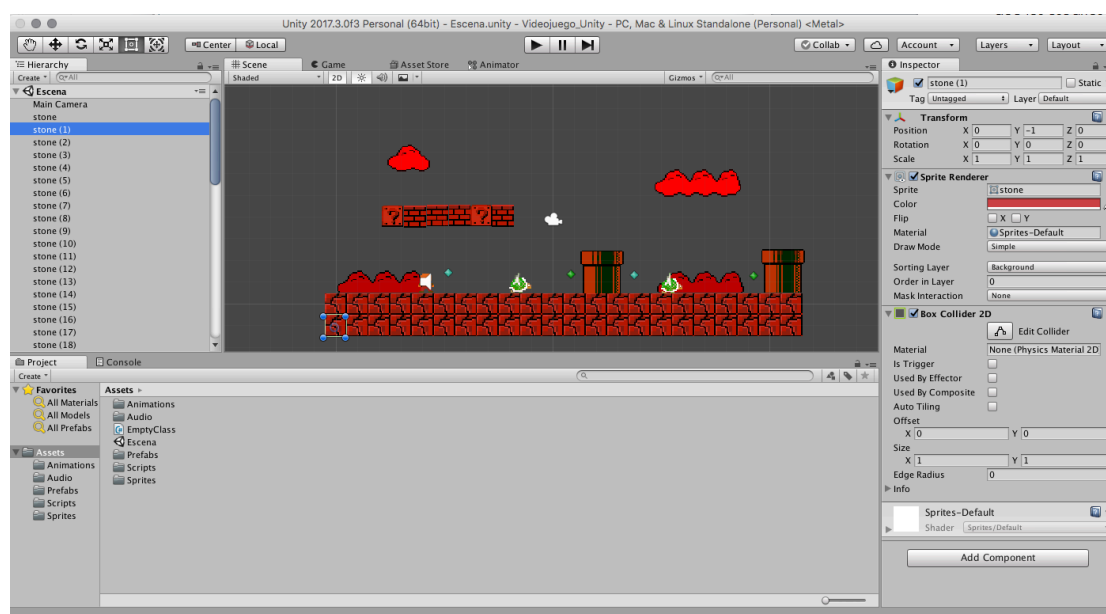
Para terminar, podemos decir que Unity es la **aplicación por excelencia en creación de juegos**. Vamos a repasar todo lo que podríamos hacer en Unity y algunas de sus características:

- **Edición:** Unity cuenta con un editor que tiene múltiples herramientas. Está disponible en Windows y Mac y permite el desarrollo en 2D y 3D.
- **Herramientas de diseño:** uno de los aspectos clave de Unity es que ofrece un espacio creativo con multitud de herramientas de gran capacidad de diseño con el que podremos realizar:
- **Renderizado de gráficos:** cuenta con un motor de renderizado en tiempo real que consigue una fidelidad visual a través de Real-Time Global Illumination y Physically Based Rendering. Además, cuenta con un API de gráficos nativos que le permite aprovechar las mejoras de GPU y hardware.
- **Plataformas:** una de las mayores ventajas que ofrece Unity en cuanto a público se refiere es su presencia en más de veinticinco plataformas (como iOS, Android, PC o Nintendo Switch, entre otras).
- **Realidad virtual y aumentada:** es la plataforma de desarrollo de VR más utilizada. Permite contar con el pipeline de renderizado y la iteración rápida del editor para crear experiencias XR de alta calidad.
- **Unity Assets Store:** tiene una tienda virtual donde se puede encontrar contenido gratuito o de pago y tiene disponible infinidad de contenidos de alta calidad.
- **Multijugador:** brinda la posibilidad de ofrecer multiplayer para que los usuarios puedan jugar en red.
- **Unity Teams:** esta prestación permite guardar, compartir y sincronizar los proyectos, así como compartir compilaciones con cualquier persona de manera automática.
- **Unity Connect:** permite que los creadores puedan exponer sus ideas y trabajos en una plataforma para poder darles salida. Del mismo modo, interconecta a gente que busca trabajo con empleadores.
- **Unity Analytics:** esta herramienta es muy útil para analizar el juego y monitorear la actividad de los jugadores.
- **Unity Performance Reporting:** genera informes de rendimiento para solucionar temas o problemas que requieren soluciones rápidas. Al mismo tiempo, recopila y reacciona errores de aplicación en dispositivos y plataformas.

- **Monetización:** mediante los análisis, Unity permite incorporar mejoras continuas que van a ser de gran ayuda a la hora de mantener la fidelización con los usuarios y, por tanto, se va a ver reflejado en la monetización del juego.

6.- Manual de usuario básico.

6.a.- Juego realizado.



Como se puede ver en la imagen anterior, el juego que se ha desarrollado es parte de uno de los niveles del famoso juego Super Mario Bros (del año 1985). Por temas legales, el personaje no puede ser realmente Super Mario, por tanto le hemos cambiado algo el aspecto y dentro del juego le hemos llamado Baby Mario.

El funcionamiento consiste básicamente en poder mover a nuestro personaje de manera horizontal, incluyendo saltos sobre los diferentes elementos del nivel (como en el juego original).

El objetivo, no es sólo demostrar que se puede programar un Video Juego en Unity sin muchos conocimiento de programación, sino también comprobar todo el potencial de los componentes anteriormente nombrados.

Se puede descargar el juego en la siguiente dirección:

https://github.com/i92ruroa/Trabajo_SLYCS/tree/master/Videojuego_Unity

[19]

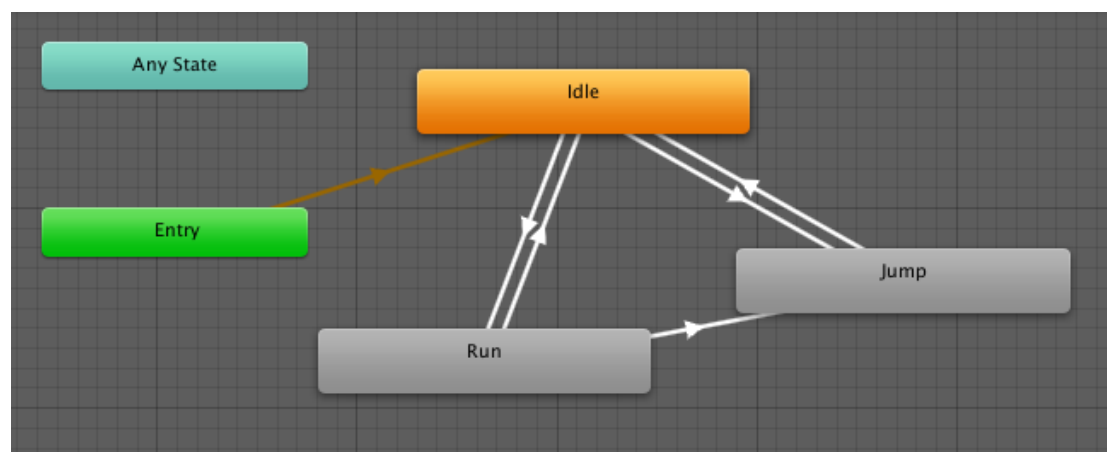
6.b.- Project Window.

Para poder llevar a cabo el funcionamiento de nuestro juego, ha sido necesario crear diferentes carpetas.

Una para las animaciones.

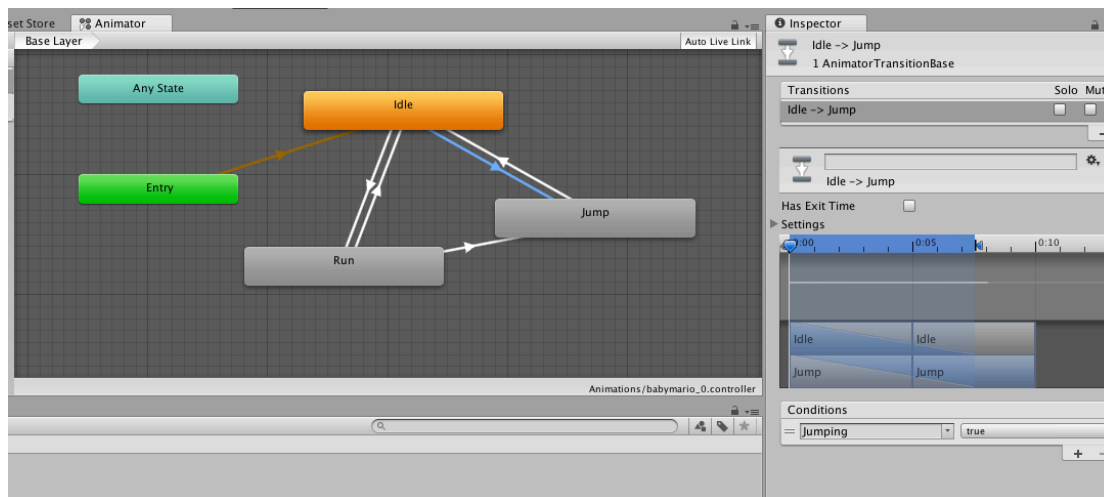
Sin entrar mucho en detalle en como trabajan las animaciones en Unity, hay que decir que simplemente es una transición entre estados.

Por ejemplo, en la siguiente imagen se puede ver una de las animaciones de nuestro juego. En la que aparecen el estado Idle (cuando nuestro personaje está quieto), corriendo o saltando. Solamente puede haber un estado como predeterminado (el que aparece en color naranja).



Sobre cada uno de estos estados, se pueden ir transiciones entre los demás (de ida o de vuelta). Y a estas transiciones, añadirle una serie de parámetros que luego podemos tener en cuenta en los Scripts de programación.

Por ejemplo, si pinchamos en la transición de estado quito a saltar, vemos que una condición llamada Jumping la hemos puesto a True.



En la contraria, la ponemos a False.

Y por ejemplo, las transiciones entre el estado Idle y corriendo, tiene en cuenta si la variable Speed es mayor o menor a una cantidad determinada por el usuario.

Otra carpeta para los sonidos

A nuestro juego le hemos añadido una serie de efectos de sonido (para el salto, background, etc), pues en esta carpeta es dónde están añadidos, para hacer uso de ellos posteriormente.

Otra carpeta para los Prefabs

Un Prefabs no es más que un elemento en Unity que se puede aprovechar de él tanto su funcionalidad como su diseño. Se hace una determinada acción una sola vez y luego se utiliza en otros elementos. Siendo importante que si se cambia en el raíz, se despliega en todos los elementos que lo han utilizado.

En nuestro caso, por ejemplo hemos hecho prefabs para cada uno de los ladrillos que vamos a pintar en nuestro juego. Si tuviéramos que dar funcionalidad uno a uno, se iría mucho tiempo y recursos.

Otra carpeta para los Scripts

Como hemos comentado, la programación en Unity está en C#. Gracias a como maneja las variables, al uso de componentes y demás, no son

necesarios muchos conocimientos de programación. A continuación vamos a mostrar el código y una breve explicación.

Box.cs: simplemente es para controlar cada una de las cajas del juego (las preguntas o los ladrillos) y comprobar cuando lo ha golpeado nuestro personaje y darle el efecto de rebote similar al que tiene en el juego.

```
using UnityEngine;
```

```
using System.Collections;
```

```
public class Box : MonoBehaviour {
```

```
    // La curva
```

```
    public AnimationCurve curve;
```

```
    // El objeto colocador
```

```
    public GameObject spawnPrefab;
```

```
    // La siguiente Caja
```

```
    public GameObject nextPrefab;
```

```
    IEnumerator sample() {
```

```
        // Posición inicial
```

```
        Vector2 pos = transform.position;
```

```
        // movimiento de la curva
```

```
        for (float t=0; t<curve.keys[curve.length-1].time; t+=Time.deltaTime) {
```

```
            transform.position = new Vector2(pos.x, pos.y + curve.Evaluate(t));
```

```
            // pasamos a la siguiente curva
```

```
            yield return null;
```

```
        }
```

```
        // Coloca la moneda
```

```
        if (spawnPrefab)
```

```
            Instantiate(spawnPrefab, transform.position + Vector3.up, Quaternion.identity);
```

```
        // Destruye la caja
```

```
        if (nextPrefab)
```

```
            Instantiate(nextPrefab, transform.position, Quaternion.identity);
```

```
        Destroy(gameObject);
```

```

    }

    void OnCollisionEnter2D(Collision2D coll) {
        // golpea desde abajo
        if (coll.contacts[0].point.y < transform.position.y)
            StartCoroutine("sample");
    }
}

```

Coin.cs: esto hace que aparezca la moneda en el lugar donde corresponda.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Coin : MonoBehaviour {

    void OnTriggerEnter2D(Collider2D coll) {
        Destroy (gameObject);
    }
}

```

Enemy.cs: en este caso es algo más complejo, y lo que se tiene en cuenta es el movimiento de cada uno de los enemigos de nuestro personaje. Moviendo a cada uno de ellos, y comprobando si se ha chocado con algún elemento del nivel del juego.

```

using UnityEngine;
using System.Collections;

public class Enemy : MonoBehaviour {
    // velocidad
    public float speed = 0.05f;
    public float jumpForce = 1200;

    // dirección actual
    Vector2 dir = Vector2.right;

    // fuerza de salto
    public float upForce = 800;
}

```

```

bool IsGrounded()
{
    // Obtener límites y rango de lanzamiento (10% de la altura)
    Bounds bounds = GetComponent<Collider2D>().bounds;
    float range = bounds.size.y * 0.1f;

    // Calcula una posición ligeramente debajo del colisionador
    Vector2 v = new Vector2(bounds.center.x,
        bounds.min.y - range);

    RaycastHit2D hit = Physics2D.Linecast(v, bounds.center);

    // comprueba y si hay algo intermedio o nos golpeamos
    return (hit.collider.gameObject != gameObject);
}

void FixedUpdate() {
    // establecemos la velocidad
    GetComponent<Rigidbody2D>().velocity = dir * speed;

    // Movimiento vertical (salto)
    bool grounded = IsGrounded();
    if (grounded)
    {
        GetComponent<Rigidbody2D>().AddForce(Vector2.up * jumpForce * 2);
        GetComponent<Animator>().SetBool("Jumping", !grounded);
    }
}

void OnTriggerEnter2D(Collider2D coll) {
    // Si golpea con algo, cambia de dirección
    transform.localScale = new Vector2(-1 * transform.localScale.x,
        transform.localScale.y);

    // Y efecto espejo
    dir = new Vector2(-1 * dir.x, dir.y);
}

void OnCollisionEnter2D(Collision2D coll) {

```



```

// Ha colisionado con nuestro personaje?
if (coll.gameObject.name == "BabyMario") {
    // Por abajo?
    if (coll.contacts[0].point.y > transform.position.y) {
        // Ejecutar animaciones
        GetComponent<Animator>().SetTrigger("Died");

        // Desahilitar el collider
        GetComponent<Collider2D>().enabled = false;

        // Empuja al personaje hacia arriba
        coll.gameObject.GetComponent<Rigidbody2D>().AddForce(Vector2.up * upForce);

        // Muere en 5 segundos
        Invoke("Die", 5);
    } else {
        // Mata al personaje
        Destroy(coll.gameObject);
    }
}

void Die() {
    Destroy(gameObject);
}
}

```

Flor.cs: esto simplemente hace que se destruya un elemento si se colisiona con él.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

```

```

public class Flor : MonoBehaviour {

    void OnTriggerEnter2D(Collider2D coll) {
        Destroy (gameObject);
    }

}

```

PlayerMovement.cs: este es similar al de los enemigos, pero aplicándolo sobre nuestro personaje. Hay que tener en cuenta el uso de GetAxis (en este caso Horizontal), que es la combinación de teclas para mover a nuestro personaje. Se puede cambiar para cualquier tipo de dispositivo. Para ello, Edit > Project Settings > Input

```
using UnityEngine;
```

```
using System.Collections;
```

```
public class PlayerMovement : MonoBehaviour
```

```
{
```

```
    AudioSource mariojump;
```

```
    // propiedades del movimiento
```

```
    public float moveSpeed = 15;
```

```
    [Range(0, 1)] public float sliding = 0.9f;
```

```
    public float jumpForce = 1200;
```

```
    void Start () {
```

```
        mariojump = GetComponent<AudioSource>();
```

```
    }
```

```
    bool IsGrounded()
```

```
    {
```

```
        Bounds bounds = GetComponent<Collider2D>().bounds;
```

```
        float range = bounds.size.y * 0.1f;
```

```
        Vector2 v = new Vector2(bounds.center.x,  
                                bounds.min.y - range);
```

```
        RaycastHit2D hit = Physics2D.Linecast(v, bounds.center);
```

```
        return (hit.collider.gameObject != gameObject);
```

```
    }
```

```
    void FixedUpdate()
```

```
    {
```

```
        // Movimiento horizontal
```

```
        float h = Input.GetAxis("Horizontal");
```

```
        Vector2 v = GetComponent<Rigidbody2D>().velocity;
```

```
        if (h != 0)
```

```
        {
```

```
            // Mover izquierda / derecha
```

```

        GetComponent<Rigidbody2D>().velocity = new Vector2(h * moveSpeed, v.y);
        transform.localScale = new Vector2(Mathf.Sign(h), transform.localScale.y);
    }
    else
    {
        // Frenar al personaje
        GetComponent<Rigidbody2D>().velocity = new Vector2(v.x * sliding, v.y);
    }
    GetComponent<Animator>().SetFloat("Speed", Mathf.Abs(h));

    // Salto
    bool grounded = IsGrounded();
    if (Input.GetKey(KeyCode.UpArrow) && grounded) {
        GetComponent<Rigidbody2D>().AddForce(Vector2.up * jumpForce);
        GetComponent<Animator>().SetBool("Jumping", !grounded);
        mariojump.Play();
    }
}

void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.tag == "seta")
    {
        transform.localScale = new Vector2(2, 2);
    }
}
}

```

Seta.cs: por último este script lo que hace es que aparezca la seta cuando el personaje colisione con el elemento correspondiente.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Seta : MonoBehaviour
{
    void OnTriggerEnter2D(Collider2D coll)
    {
        Destroy(gameObject);
    }
}

```

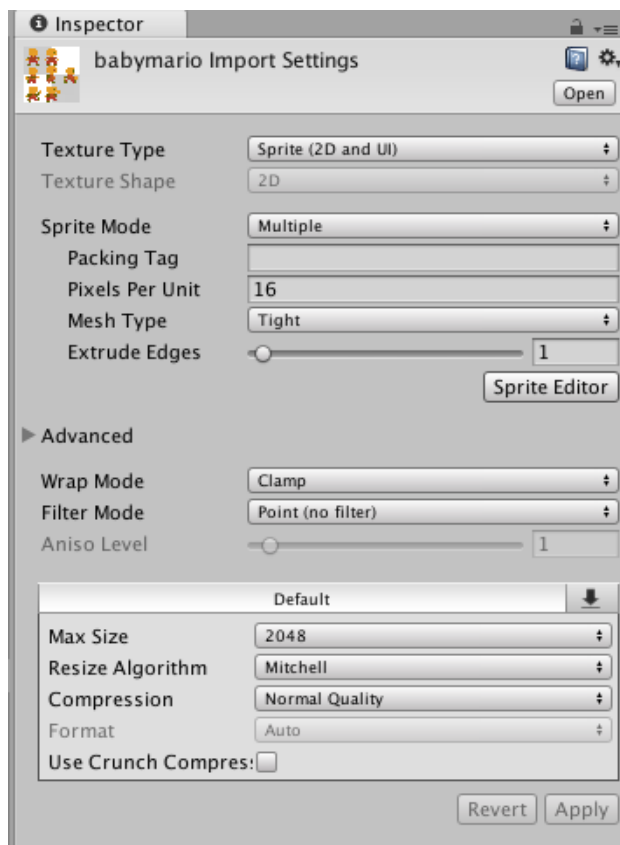
}

Y por último, otra carpeta para los Sprites

Aquí, pueden aparecer bien imágenes individuales, o Sprites como tal. Un Sprite es una imagen en la que están añadidos cada uno de los movimientos de un personaje. En la siguiente imagen se puede ver un ejemplo de Sprite para uno de los personajes de Street Fighter II.



Unity tiene la capacidad (no como otros Softwares para el desarrollo de Videojuegos), de detectar automáticamente cada uno de los elementos de nuestro Sprite.



Para poder aplicar esto, nuestro sprite debe estar como Múltiple. Al hacer esto, y clickar en Sprite Editor vemos que se identifica claramente cada uno de los movimientos.

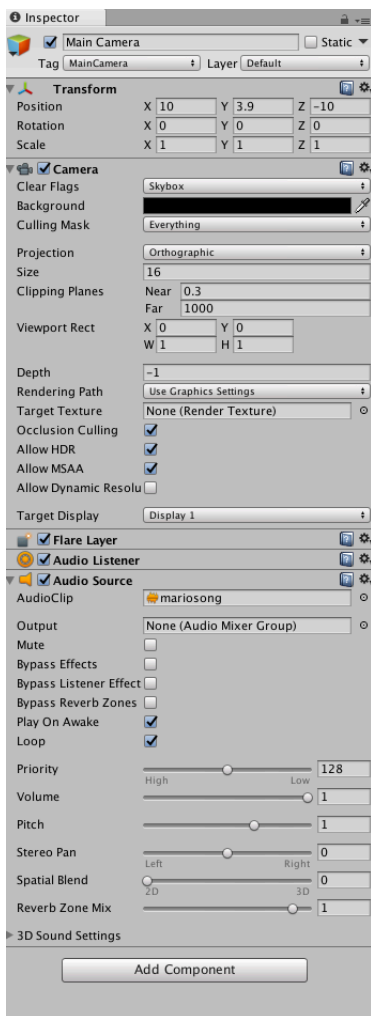
Aprovechando esta imagen, hay que tener en cuenta los Píxeles por Unidad. Es recomendable que todos los elementos de nuestro juego tengan unos valores proporcionados.

6.c.- Hierarchy Window + Inspector Window

Cómo hemos comentado anteriormente, en esta parte del programa es dónde aparecen todos los elementos que van a formar parte de nuestro juego. En nuestro caso, la cámara principal, cada una de las piedras del suelo, las nubes, las tuberías, nuestro personaje, las interrogaciones, etc.

Es desde la ventana de jerarquía, desde donde puedes ponerlos en la posición que el usuario desee y darle las propiedades necesarias.

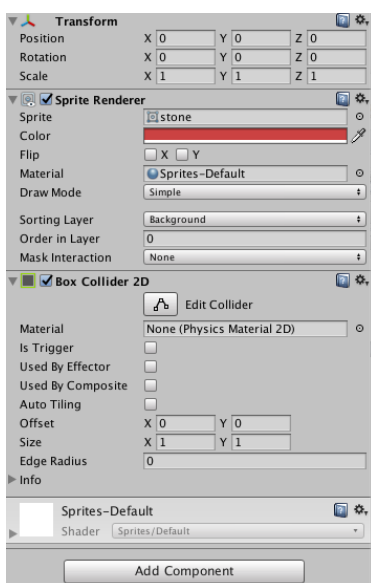
Vamos a ver algunos de estos elementos, los más importantes.



Main Camera

Al pulsar, aparece en el inspector detalles como la posición, la rotación, el tipo de fondo, el tamaño (relacionado con los píxeles por unidad de las imágenes), la profundidad (los elementos se van colocando por capas, estando más abajo los de un valor inferior).

Aquí también se añaden otros elementos, como son los sonidos de fondo.

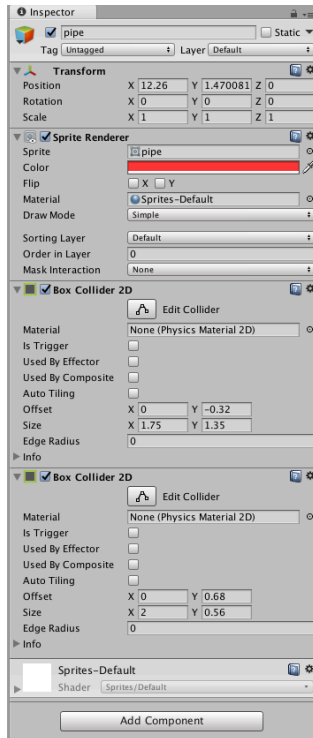


Stone (cada una de las rocas)

Aquí también se puede ver la posición, rotación y escala.

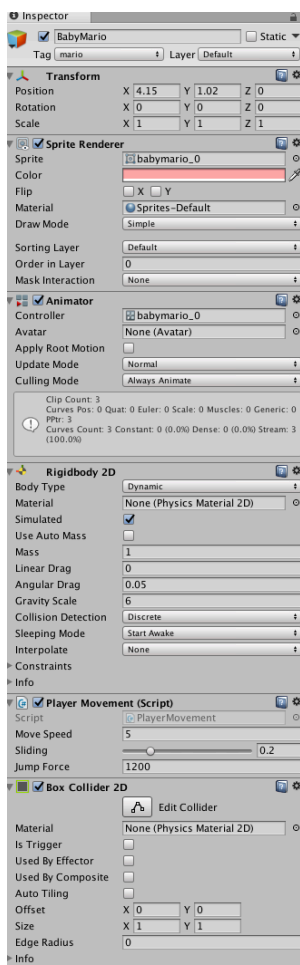
Características del Sprite, como el color o posición dentro de nuestra interfaz del juego.

Y como muchos de los Game Object de este juego, le hemos asignado el componente Box Collider 2D. Con esa simple asignación, con este elemento ya podrá colisionar nuestro personaje y los enemigos.



Pipe (cada una de las dos tuberías)

Aquí hay una pequeña diferencia con el caso anterior, ya que hemos asignado dos Box Collider 2D, una para la parte baja de la tubería, y otro para la parte superior.



BabyMario (nuestro personaje)

En este caso es mucho más complejo que los Game Object anteriores.

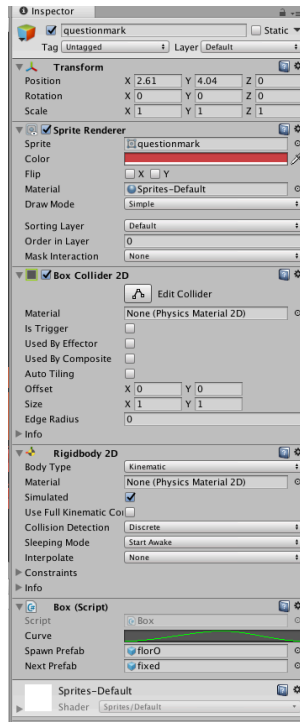
Le hemos asignado una de las animaciones antes comentadas.

También le hemos asignado el componente Rigidbody 2D, para que el personaje tenga "física" y se pueda mover.

Le hemos asignado el Script Player Movement (ya visto), con unas variables dinámicas establecidas en el código (se puede cambiar bien desde código o desde el inspector).

Le hemos asignado Box Collider 2D, para que pueda colisionar.

Y por último, le hemos asignado los componentes de audio, para cada vez que salta o colisiona con algún elemento, asociarle unos efectos de sonido de los que ya hemos hablado.

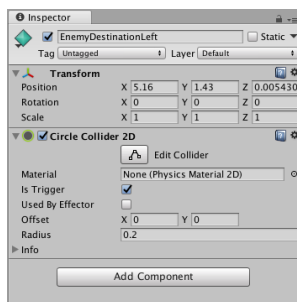


QuestionMark (las interrogaciones)

En este caso le hemos asignado Box Collider, para que puedan colisionar con él.

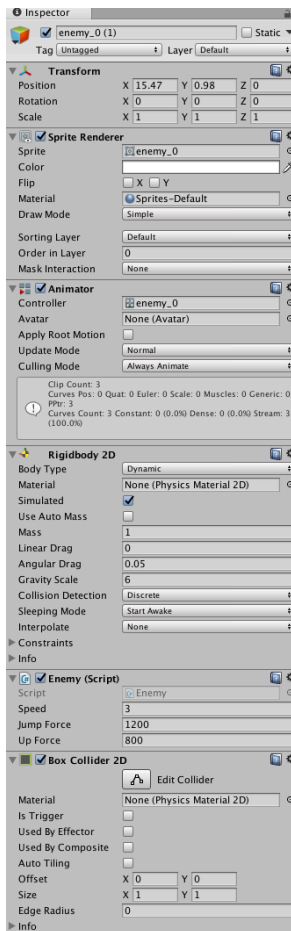
También Rigidbody 2D, porque cada vez que el personaje colisiona, se mueve.

Y por último, el Script Box, para darle la curva de movimiento (esto también se aplica a cada uno de los **Bricks** (ladrillos) del juego).



Enemy Destination Left / Right (enemigo que va saltando)

En este caso le hemos asignado Circle Collider 2D, para que el movimiento sea circular.



Enemy_0 (enemigo que va de izquierda a derecha)

Por último, está el Game Object asociado al enemigo que va de izquierda a derecha.

Al igual que Baby Mario, también tiene asociada una de las animaciones, Rigidbody 2D y Box Collier 2D, y en este caso asociado el Script Enemy con las variables establecidas desde código que también se pueden cambiar aquí.

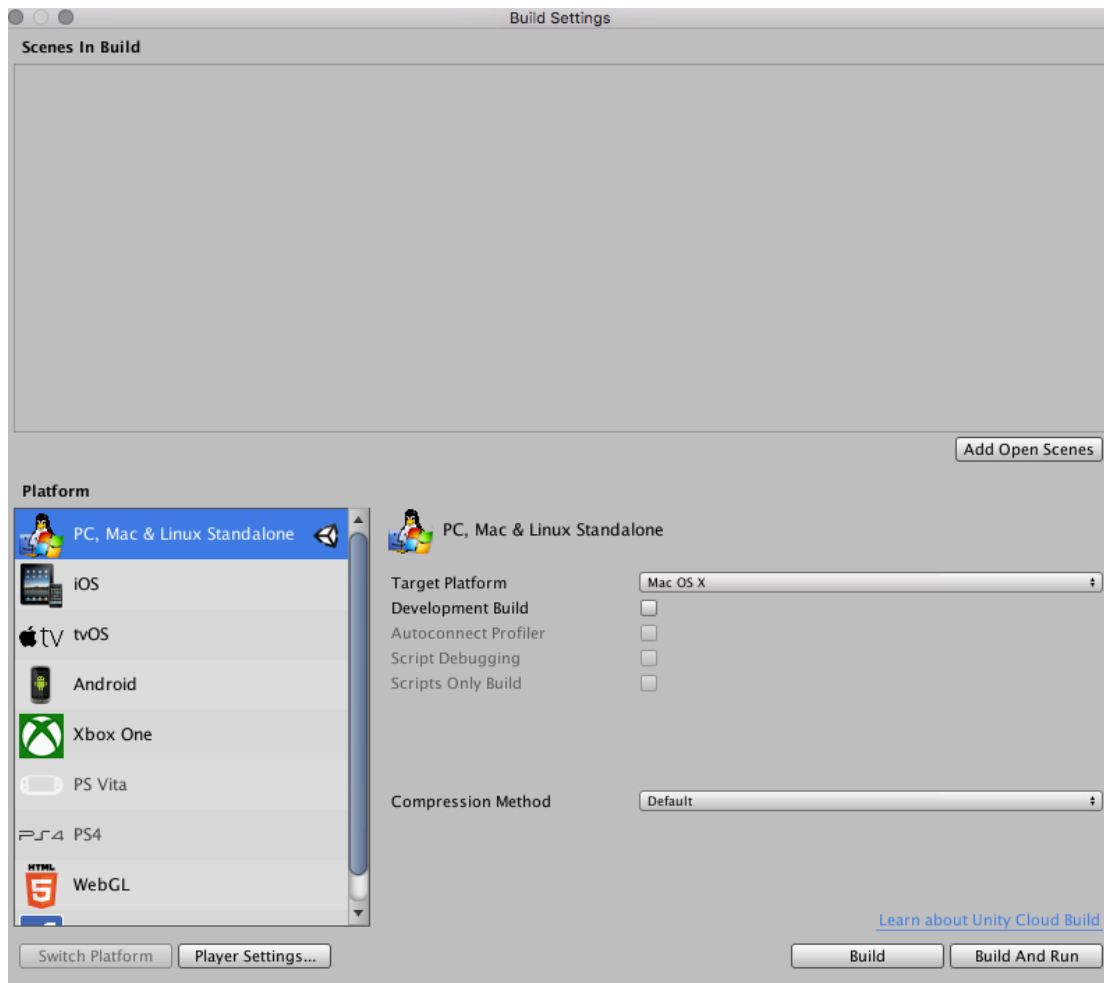
En definitiva, como hemos podido ver en este pequeño ejemplo, trabajar con Unity es muy fácil. Con apenas conocimientos de programación, podemos crear nuestros propios Scripts, asociarlos a los Game Objects y poco a poco, ir dándole forma a la idea que tenemos en mente.

6.d.- Ejecutando el juego

Bien, ya que tenemos nuestro juego desarrollado y depurado, es hora de poder jugar con él.

Como hemos comentado anteriormente, una de las propiedades más significativas de Unity es que es un software que hace que los juegos sean multidispositivo.

Así, para crear el ejecutable del juego hay que irse a File > Build Settings



Como se puede ver en la imagen anterior, se puede ejecutar para los siguientes dispositivos:

- PC, Mac o Linux.
- APP para IOS (para lo que hace falta tener licencia de desarrollador e instalar una serie de módulos).
- TV para IOS (igual que el caso anterior).
- Android (en este caso únicamente hace falta instalar una serie de módulos, y ya decidir si el juego será gratuito o de pago)
- Xbox, PS One o Playstation 4 (en este caso, aunque supuestamente sólo hay que instalar unos módulos, las compañías de estas consolas si piden una serie de pagos para poder generar el ejecutable).
- Facebook (también se puede crear los típicos juegos de Facebook; pero al igual que el caso anterior, es necesaria una suma de dinero bastante importante).

En nuestro caso, ya que trabajamos sobre Mac, seleccionamos la primera opción (con la configuración que se desee).

Al final, pinchando en el archivo que se genera (o en el exe que se genera en Windows), ya podemos disfrutar de nuestro juego.

7.- Extra. Otros juegos con Software Libre.

7.a.- Phaser.

Ya hemos visto que Unity es uno de los Software más potentes para el desarrollo de Videojuegos.

Pero cómo desarrollador Web que soy, y con amplia experiencia en Javascript, quiero demostrar que con este lenguaje de programación se puede llevar a cabo el desarrollo de un Video Juego, teniendo siempre muy clara cual va a ser la metodología del mismo.

El primer ejemplo que vamos a ver está desarrollado en Phaser (<https://phaser.io/>) [12]. Es un Framework de Javascript creado exclusivamente para el desarrollo de Videojuegos.

Actualmente, la última versión estable es la 3.23.0, y para poder usarlo tan sólo hay que descargarse el código de Github e integrarlo en nuestro código.

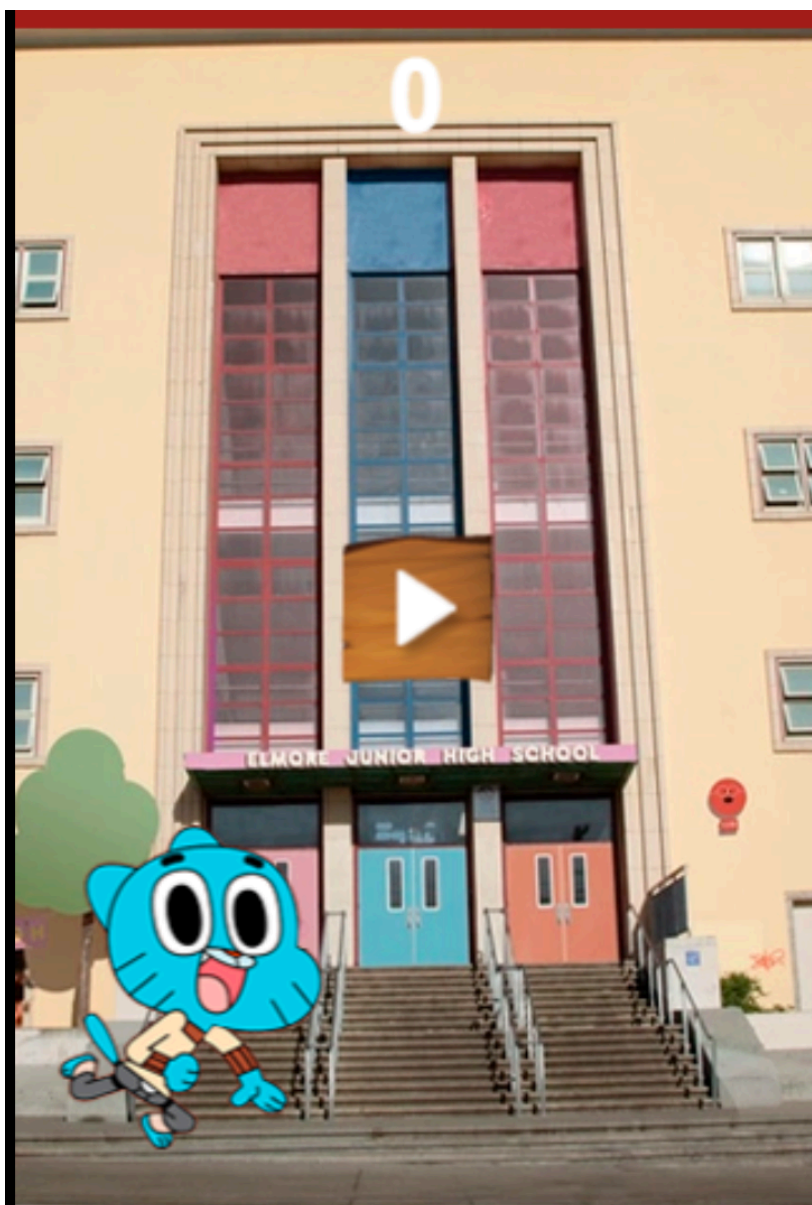
Hay una comunidad bastante grande detrás de Phaser. Hay muchos tutoriales, foros de ayuda y ejemplos de código que pueden ayudar a los que quieren adentrarse en este mundo.

En nuestro caso, hemos desarrollado un simple juego en el que van cayendo objetos desde el cielo, y el protagonista tiene que esquivarlos en un determinado tiempo.

Nos hemos basado en los personajes del Maravilloso Mundo de Gumball (dibujos animados preferidos de mi hijo de 2 años, ese el motivo).

El juego se puede descargar en https://github.com/i92ruroa/Trabajo_SLYCS/tree/master/Videojuego_Phaser [20] y se puede jugar online en la siguiente dirección:

https://www.leketembe.com/SLYCS/Videojuego_Phaser/ [21]



Con poco más de 200 líneas de código se ha podido programar el Video Juego.

El archivo principal, y que aparece directamente en el Navegador, es el index.html. Que lo único que hace es llamar al código de Phaser que hay

que descargarse de la página oficial, y también llamar al código Javascript en dónde viene toda la filosofía de nuestro juego.

```
<script src="src/phaser.min.js"></script>
<script src="src/GamePlay.js"></script>
```

Vamos a pasar directamente al código `GamePlay.js`, ya que nos hemos descargado en este caso la versión min del Framework y tampoco hay que entrar mucho en detalle de cómo está estructurada.

Como hemos comentado, hay muchos tutoriales de Phaser. Al ser Javascript, quién sepa programar en este lenguaje se adaptará rápido al Framework. Tan sólo hay que tener en cuenta que hay 3 zonas importantes en el código:

- El método **preload**, que es dónde se declaran todas las variables importantes del juego y es donde se inicializa todos los elementos necesarios para que se pueda desarrollar (carga de imágenes, de sonidos, de sprites, etc)
- El método **create**, que es el principal del juego y donde se da la funcionalidad del mismo.
- Y el método **update**, que se ejecuta una vez cada fotograma (cuya duración podemos cambiar mediante código).

Como podremos ver en el siguiente código, con estos 3 métodos, y algunas funciones auxiliares, se puede programar sin ningún problema.

```
var STATE_GAME_NONE      = 0;
var STATE_GAME_LOADING   = 1;
var STATE_GAME_PLAYING   = 2;
var STATE_GAME_GAME_OVER = 3;
var STATE_GAME_WIN       = 4;

var stateGame = STATE_GAME_NONE;

var distanceTrunks = 70;

GamePlayManager = {
  init: function() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
```

```

game.scale.pageAlignVertically = true;

this.cursors = game.input.keyboard.createCursorKeys();
this.pressEnable = true;
},
preload: function() {
    stateGame = STATE_GAME_LOADING;

    game.load.image('background','assets/images/background.png');
    game.load.image('man_stand', 'assets/images/man_stand.png');
    game.load.image('man_hit', 'assets/images/man_hit.png');
    game.load.image('trunk', 'assets/images/trunk.png');
    game.load.image('trunk2', 'assets/images/trunk2.png');
    game.load.image('trunk3', 'assets/images/trunk3.png');
    game.load.image('tomb', 'assets/images/tomb.png');

    game.load.audio('loopMusic', 'assets/sounds/musicLoop.mp3');
    game.load.audio('sfxHit', 'assets/sounds/sfxHit.mp3');
    game.load.audio('sfxGameOver', 'assets/sounds/sfxGameOver.mp3');

    game.load.spritesheet('buttonPlay', 'assets/images/buttonPlay.png', 65, 65, 2);
},
create: function() {
    this.sequence = [];

    game.add.sprite(0,0,'background');
    this.man = game.add.sprite(80, 460, 'man_stand');
    this.man.anchor.setTo(0.5, 1);
    this.buttonPlay = game.add.button(game.width/2 , game.height/2, 'buttonPlay', this.startGame, this, 1,
0, 1, 0);
    this.buttonPlay.anchor.setTo(0.5);

    //Trunks
    this.trunks = game.add.group();
    for(var i=0; i<30; i++){
        var number = game.rnd.integerInRange(0, 2);
        if(number == 0)
            var trunk = this.trunks.create(0, 0, 'trunk');
        else if(number == 1)
            var trunk = this.trunks.create(0, 0, 'trunk2');
        else

```

```

        var trunk = this.trunks.create(0, 0, 'trunk3');
        trunk.anchor.setTo(0, 0.5);
        trunk.kill();
    }

    this.tomb = game.add.sprite(80, 460, 'tomb');
    this.tomb.anchor.setTo(0.5, 1);
    this.tomb.visible = false;

    var style = {
        font: 'bold 30pt Arial',
        fill: '#FFFFFF',
        align: 'center'
    }

    this.currentScore = 0;
    this.textfield = game.add.text(game.width/2, 40, this.currentScore.toString(), style);
    this.textfield.anchor.setTo(0.5);

    var pixel = game.add.bitmapData(1,1);
    pixel.ctx.fillStyle = '#A0221E';
    pixel.ctx.fillRect(0,0,1,1);

    this.bar = game.add.sprite(0,0,pixel);
    this.bar.anchor.setTo(0);
    this.bar.width = game.width;
    this.bar.height = 10;

    this.sfxGameOver = game.add.audio('sfxGameOver');
    this.sfxHit = game.add.audio('sfxHit');
    this.loopMusic = game.add.audio('loopMusic');

    },
    refreshBar:function(value){
        var newWidth = this.bar.width + value;
        if(newWidth>game.width){
            newWidth = game.width;
        }
        if(newWidth<=0){
            newWidth = 0;
            this.gameOver();
        }
    }
}

```

```

    }
    this.bar.width = newWidth;
},
startGame: function(){
    stateGame = STATE_GAME_PLAYING;
    this.buttonPlay.visible = false;

    this.loopMusic.loop = true;
    this.loopMusic.play();

    this.bar.width = game.width;

    for(var i=0; i<this.sequence.length; i++){
        if(this.sequence[i]!=null){
            this.sequence[i].kill();
        }
    }

    this.currentScore = 0;
    this.textfield.text = this.currentScore.toString();

    this.sequence = [];
    this.sequence.push(null);
    this.sequence.push(null);
    this.sequence.push(null);

    this.man.visible = true;
    this.tomb.visible = false;

    for(var i=0; i<10; i++){
        this.addTrunk();
    }
    this.printSequence();
},
increaseScore:function(){
    this.currentScore += 100;
    this.textfield.text = this.currentScore.toString();
},
hitMan:function(direction){
    this.sfxHit.play();

```



```

for(var i=0; i<this.sequence.length; i++){
    if(this.sequence[i]!=null){
        this.sequence[i].y+=distanceTrunks;
    }
}

var firstTrunk = this.sequence.shift();
if(firstTrunk!=null){
    firstTrunk.kill();
}

this.addTrunk();

var checkTrunk = this.sequence[0];
if(checkTrunk!=null && checkTrunk.direction == direction){
    this.gameOver();
}else{
    this.increaseScore();
}

this.printSequence();
},
gameOver:function(){
    stateGame = STATE_GAME_GAME_OVER;

    this.loopMusic.stop();
    this.sfxGameOver.play();

    this.man.visible = false;
    this.tomb.visible = true;
    this.tomb.x = this.man.x;

    this.buttonPlay.visible = true;
},
addTrunk:function(){
    this.refreshBar(6);

    var number = game.rnd.integerInRange(-1, 1);
    if(number==1){

```

```

    var trunk = this.trunks.getFirstDead();
    trunk.direction = 1;
    trunk.scale.setTo(1,1);
    trunk.reset(game.world.centerX, 380 - (this.sequence.length) * distanceTrunks);
    this.sequence.push(trunk);
} else if(number===-1){

    var trunk = this.trunks.getFirstDead();
    trunk.direction = -1;
    trunk.scale.setTo(-1,1);
    trunk.reset(game.world.centerX, 380 - (this.sequence.length) * distanceTrunks);
    this.sequence.push(trunk);
} else {

    this.sequence.push(null);
}
},
printSequence: function() {
    var stringSequence = "";
    for(var i=0; i<this.sequence.length; i++){
        if(this.sequence[i]==null){
            stringSequence += "0,";
        } else {
            stringSequence += this.sequence[i].direction+",";
        }
    }
    console.log(stringSequence);
},
update: function() {
    switch(stateGame){
        case STATE_GAME_NONE:

            break;
        case STATE_GAME_LOADING:

            break;
        case STATE_GAME_PLAYING:
            this.refreshBar(-0.5);

            if(this.cursors.left.isDown && this.pressEnable){

```

```

        this.pressEnable = false;
        this.man.x = 80;
        this.man.scale.setTo(1, 1);
        this.man.loadTexture('man_hit');
        this.hitMan(-1);
    }

    if(this.cursors.right.isDown && this.pressEnable){
        this.pressEnable = false;
        this.man.x = 240;
        this.man.scale.setTo(-1, 1);
        this.man.loadTexture('man_hit');
        this.hitMan(1);
    }

    if(this.cursors.left.isUp && this.cursors.right.isUp){
        this.pressEnable = true;
        this.man.loadTexture('man_stand');
    }
    break;
case STATE_GAME_GAME_OVER:
    console.log("GAME OVER");
    break;
case STATE_GAME_WIN:

    break;
    }
}
}

```

```

var game = new Phaser.Game(320, 480, Phaser.AUTO);

```

```

game.state.add("gameplay", GameManager);
game.state.start("gameplay");

```

Aparte de esto, hay una carpeta llamada assets dónde hemos colocado nuestras imágenes y nuestros sonidos.

Hemos visto lo fácil que es programar un Video Juego en Phaser.

7.b.- Javascript + Html5.

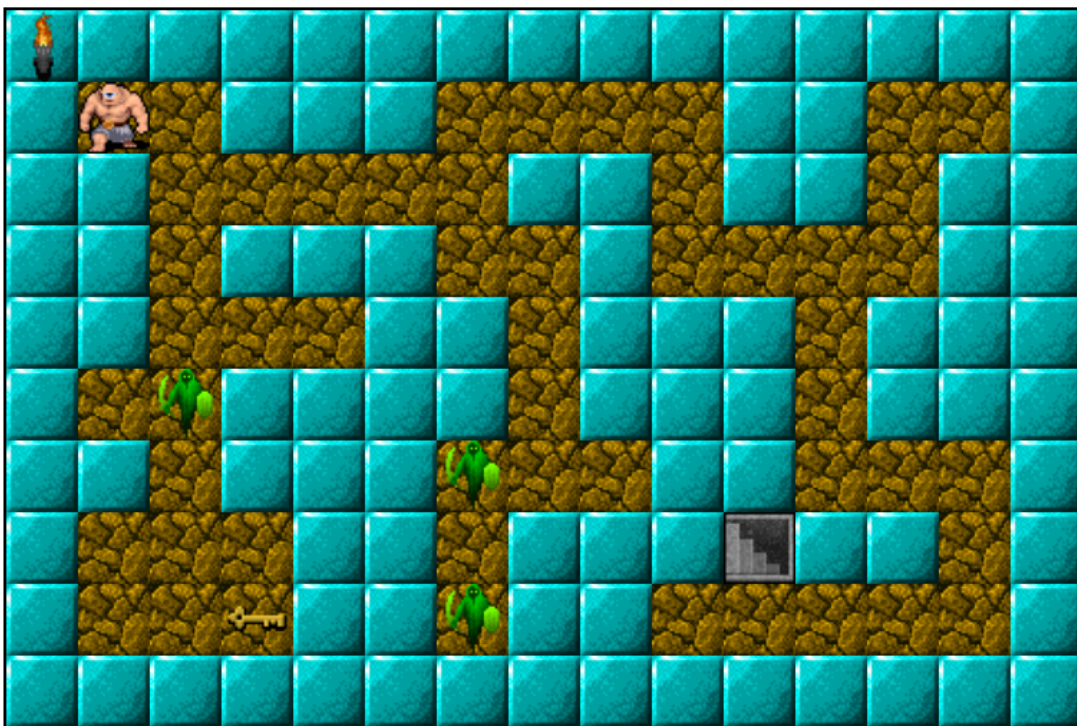
Para terminar, hemos querido demostrar que con Javascript y Html5 (puro y duro) también se puede desarrollar un Videojuego.

El juego consiste en un laberinto (hecho a través de Sprites, como en Unity, pero programándolo en forma de una matriz), en el que van moviéndose de manera aleatoria una serie de enemigos, y nuestro personaje debe ir de un punto A a otro B sin encontrarse en su camino con estos enemigos, obteniendo para ello una llave que colocamos en nuestro tablero de juego.

El juego se puede descargar en

https://github.com/i92ruoa/Trabajo_SLYCS/tree/master/Videojuego_Javascript [22] y se puede jugar online en la siguiente dirección:

https://www.leketembe.com/SLYCS/Videojuego_Javascript/ [23]



El código tan solo consiste en un archivo index.html, en el que se llama al código javascript de nuestro desarrollo, y en el que se establece un canvas (que será la zona en que se establecerán todos los elementos del juego)

```

<html>

<head>
  <title>Videojuego en Javascript - Antonio Ruiz Roldan</title>
  <script src='js/juego.js'></script>
</head>

<body onload='inicializa();'>
  <canvas id='canvas' width='750' height='500' style='border:2px solid #000000;'></canvas>
</body>

</html>

```

Hay una carpeta, en la que guardamos las imágenes (en forma de Sprite), y un código Javascript (de poco más de 350 líneas de código) super fácil de entender. Usando funciones, bucles, iteraciones y poco más creamos nuestro juego.

```

var canvas;
var ctx;
var FPS = 50;

var anchoF = 50;
var altoF = 50;

var muro = '#044f14';
var puerta = '#3a1700';
var tierra = '#c6892f';
var llave = '#c6bc00';

var protagonista;

var enemigo=[];

var imagenAntorcha;

var tileMap;

var escenario = [

```

```

[0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,2,2,0,0,0,2,2,2,0,0,2,2,0],
[0,0,2,2,2,2,0,0,2,0,0,2,0,0],
[0,0,2,0,0,0,2,2,0,2,2,2,0,0],
[0,0,2,2,0,0,2,0,0,0,2,0,0,0],
[0,2,2,0,0,0,0,2,0,0,0,2,0,0,0],
[0,0,2,0,0,0,2,2,2,0,0,2,2,2,0],
[0,2,2,0,0,2,0,0,0,1,0,0,2,0],
[0,2,2,3,0,0,2,0,0,2,2,2,2,2,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
]

```

```
function dibujaEscenario(){
```

```

    for(y=0;y<10;y++){
        for(x=0;x<15;x++){

            var tile = escenario[y][x];
            ctx.drawImage(tileMap,tile*32,0,32,32,anchoF*x,altoF*y,anchoF,altoF);
        }
    }
}

```

```

var antorcha = function(x,y){
    this.x = x;
    this.y = y;

```

```

    this.retraso = 10;
    this.contador = 0;
    this.fotograma = 0; //0-3

```

```

    this.cambiaFotograma = function(){
        if(this.fotograma < 3) {
            this.fotograma++;
        }
        else{
            this.fotograma = 0;
        }
    }
}

```

```
}
```

```
this.dibuja = function(){
```

```
    if(this.contador < this.retraso){
```

```
        this.contador++;
```

```
    }
```

```
    else{
```

```
        this.contador = 0;
```

```
        this.cambiaFotograma();
```

```
    }
```

```
    ctx.drawImage(tileMap,this.fotograma*32,64,32,32,anchoF*x,altoF*y,anchoF,altoF);
```

```
}
```

```
}
```

```
//CLASE ENEMIGO
```

```
var malo = function(x,y){
```

```
    this.x = x;
```

```
    this.y = y;
```

```
    this.direccion = Math.floor(Math.random()*4);
```

```
    this.retraso = 50;
```

```
    this.fotograma = 0;
```

```
    this.dibuja = function(){
```

```
        ctx.drawImage(tileMap,0,32,32,32,this.x*anchoF,this.y*altoF,anchoF,altoF);
```

```
    }
```

```
    this.compruebaColision = function(x,y){
```

```
        var colisiona = false;
```

```

    if(escenario[y][x]==0){
        colisiona = true;
    }
    return colisiona;
}

```

```

this.mueve = function(){

```

```

    protagonista.colisionEnemigo(this.x, this.y);

```

```

    if(this.contador < this.retraso){
        this.contador++;
    }

```

```

    else{
        this.contador = 0;

```

```

//ARRIBA

```

```

    if(this.direccion == 0){
        if(this.compruebaColision(this.x, this.y - 1)==false){
            this.y--;
        }
        else{
            this.direccion = Math.floor(Math.random()*4);
        }
    }
}

```

```

//ABAJO

```

```

    if(this.direccion == 1){
        if(this.compruebaColision(this.x, this.y + 1)==false){
            this.y++;
        }
        else{
            this.direccion = Math.floor(Math.random()*4);
        }
    }
}

```

```

//IZQUIERDA

```



```

if(this.direccion == 2){
    if(this.compruebaColision(this.x - 1, this.y)==false){
        this.x--;
    }
    else{
        this.direccion = Math.floor(Math.random()*4);
    }
}

//IZQUIERDA
if(this.direccion == 3){
    if(this.compruebaColision(this.x + 1, this.y)==false){
        this.x++;
    }
    else{
        this.direccion = Math.floor(Math.random()*4);
    }
}
}

}

//OBJETO JUGADOR
var jugador = function(){
    this.x = 1;
    this.y = 1;
    this.color = '#820c01';
    this.llave = false;

    this.dibuja = function(){
        ctx.drawImage(tileMap,32,32,32,32,this.x*anchoF,this.y*altoF,anchoF,altoF);
    }

    this.colisionEnemigo = function(x,y){
        if(this.x == x && this.y == y){
            this.muerte();
        }
    }
}

```

```
}
```

```
}
```

```
this.margenes = function(x,y){  
    var colision = false;
```

```
    if(escenario[y][x]==0){  
        colision = true;  
    }
```

```
    return(colision);  
}
```

```
this.arriba = function(){  
    if(this.margenes(this.x, this.y-1)==false){  
        this.y--;  
        this.logicaObjetos();  
    }  
}
```

```
this.abajo = function(){  
    if(this.margenes(this.x, this.y+1)==false){  
        this.y++;  
        this.logicaObjetos();  
    }  
}
```

```
this.izquierda = function(){  
    if(this.margenes(this.x-1, this.y)==false){  
        this.x--;  
        this.logicaObjetos();  
    }  
}
```

```
this.derecha = function(){  
    if(this.margenes(this.x+1, this.y)==false){
```

```

    this.x++;
    this.logicaObjetos();
  }
}

this.victoria = function(){
  console.log('Has ganado!');

  this.x = 1;
  this.y = 1;

  this.llave = false; //el jugador ya no tiene la llave
  escenario[8][3] = 3; //volvemos a poner la llave en su sitio
}

this.muerte = function(){
  console.log('Has perdido!');

  this.x = 1;
  this.y = 1;

  this.llave = false; //el jugador ya no tiene la llave
  escenario[8][3] = 3; //volvemos a poner la llave en su sitio
}

this.logicaObjetos = function(){
  var objeto = escenario[this.y][this.x];

  //OBTIENE llave
  if(objeto == 3){
    this.llave = true;
    escenario[this.y][this.x]=2;
    console.log('Has obtenido la llave!!');
  }
}

```

```

//ABRIMOS LA PUERTA
if(objeto == 1){
  if(this.llave == true)
    this.victoria();
  else{
    console.log('No tienes la llave, no puedes pasar!');
  }
}
}
}

```

```

function inicializa(){
  canvas = document.getElementById('canvas');
  ctx = canvas.getContext('2d');

```

```

  tileMap = new Image();
  tileMap.src = 'img/tilemap.png';

```

```

//CREAMOS AL JUGADOR
protagonista = new jugador();

```

```

//CREAMOS LA antorcha
imagenAntorcha = new antorcha(0,0);

```

```

//CREAMOS LOS ENEMIGOS
enemigo.push(new malo(3,3));
enemigo.push(new malo(5,7));
enemigo.push(new malo(7,7));

```

```

//LECTURA DEL TECLADO
document.addEventListener('keydown',function(tecla){

```

```

    if(tecla.keyCode == 38){
        protagonista.arriba();
    }

    if(tecla.keyCode == 40){
        protagonista.abajo();
    }

    if(tecla.keyCode == 37){
        protagonista.izquierda();
    }

    if(tecla.keyCode == 39){
        protagonista.derecha();
    }

});

setInterval(function(){
    principal();
}, 1000/FPS);
}

function borraCanvas(){
    canvas.width=750;
    canvas.height=500;
}

function principal(){
    borraCanvas();
    dibujaEscenario();
    imagenAntorcha.dibuja();
    protagonista.dibuja();

    for(c=0; c<enemigo.length; c++){
        enemigo[c].mueve();
        enemigo[c].dibuja();
    }
}

```

}

8.- Referencias.

[1] <https://www.tartessosinternacional.com/animaciones-3d-juegos-entornos-interactivos/>

[2] <https://pl4yers.com/2018/11/la-industria-de-los-videojuegos-su-consumo-y-evolucion/>

[3] <https://unity.com>

[4] <https://unity.com/es/education/license-grant-program>

[5] <https://www.scirra.com/>

[6] <https://www.yoyogames.com/>

[7] <https://godotengine.org/>

[8] <https://www.unrealengine.com/en-US/>

[9] <http://www.stencyl.com/>

[10] <https://gdevelop-app.com/>

[11] <https://defold.com/>

[12] <https://phaser.io/>

[13] <https://www.unityspain.com/>

[14] <https://unity.com/es/community>

[15] <https://unity3d.com/es/get-unity/download/archive>

[16] <https://id.unity.com/en/conversations/e3d5ac12-6b75-46df-bb5d-d2c98f3004b2007f>

[17] <https://docs.unity3d.com/es/530/Manual/UISystem.html>

[18] <https://docs.unity3d.com/es/2018.4/Manual/GameObjects.html>

[19]
https://github.com/i92ruroa/Trabajo_SLYCS/tree/master/Videojuego_Unity

[20]
https://github.com/i92ruroa/Trabajo_SLYCS/tree/master/Videojuego_Phaser
r

[21] https://www.leketembe.com/SLYCS/Videojuego_Phaser/

[22]
https://github.com/i92ruroa/Trabajo_SLYCS/tree/master/Videojuego_Javascript
cript

[23] https://www.leketembe.com/SLYCS/Videojuego_Javascript/