# Memoria en formato PDF

* [Memoria en formato PDF](#memoria-en-formato-pdf)
  + [Descripción del videojuego](#descripci%c3%b3n-del-videojuego)
  + [Herramientas utilizadas](#herramientas-utilizadas)
  + [Diseño/Arquitectura del software (a ser posible con gráficos que indiquen relaciones: diagramas de clase, de comportamiento,…) en el que figuren los componentes de la aplicación:](#X83f8672c434728c72310917f9be9ef3fe5d5c4d)
    - [Ficheros](#ficheros)
      * [Animaciones](#animaciones)
      * [Gráficos (Sprites)](#gr%c3%a1ficos-sprites)
      * [Materiales](#materiales)
      * [Escenas](#escenas)
    - [Módulos y clases](#m%c3%b3dulos-y-clases)
      * [Controlador de la cámara](#controlador-de-la-c%c3%a1mara)
      * [Controlador del personaje](#controlador-del-personaje)
      * [Controlador de enemigos](#controlador-de-enemigos)
      * [Controlador de la puerta](#controlador-de-la-puerta)
      * [Controlador de la plataforma que pierde bloques](#X3b3815f5540a21e4a38acbf700a1e88f7557d42)
      * [Elementos que causan daño (roca)](#elementos-que-causan-da%c3%b1o-roca)
      * [Menús](#men%c3%bas)
    - [Escenas](#escenas-1)
    - [Actores](#actores)
    - [Recursos:](#recursos)
  + [Planificación](#planificaci%c3%b3n)
  + [Temporización real](#temporizaci%c3%b3n-real)
  + [Valoración de los resultados](#valoraci%c3%b3n-de-los-resultados)
  + [Defectos](#defectos)
  + [Bibliografía/Webgrafía](#bibliograf%c3%adawebgraf%c3%ada)

## Descripción del videojuego

Este es un videojuego en dos dimensiones de los llamados “de plataformas”. Consiste en alcanzar una puerta al siguiente nivel avanzando por un nivel con plataformas, trampas y enemigos a los que evitar. \* Plataforma del videojuego: Este juego ha sido desarrollado en Unity, por lo que se puede implementar en varias plataformas sin necesidad de realizar una gran cantidad de modificaciones, a pesar de ello su desarrollo ha sido orientado hacia PC y se compilará para Windows. \* Mecánica del videojuego: Para avanzar en el videojuego deberemos recorrer cada nivel buscando una puerta que nos llevará al nivel siguiente. En algunos niveles la puerta estará bloqueada y necesitaremos encontrar un interruptor que la desbloquee, en otros casos la puerta estará desbloqueada pero en el camino nos encontraremos con enemigos o plataformas trampa u ocultas. Consideraremos que nos habremos pasado el videojuego cuando acabemos el último nivel y volvamos a la pantalla de inicio. \* Manejo por parte del usuario: El personaje dispone de los controles básicos para moverse, atacar y saltar, una o varias veces seguidas dependiendo del nivel. Al ser un juego en 2D y no tener profundidad, el personaje solo se moverá hacia izquierda o derecha, y lo controlaremos con las flechas o con A y D (A para izquierda y D para derecha). Para saltar se presionará la flecha arriba o la letra W. En el caso de que el nivel permita saltar varias veces seguidas (en el aire), la cantidad de saltos se reestablecerá cada vez que se toque el suelo o una plataforma. Cada vez que se ataque, presionando la tecla Espacio, el personaje lanzará una roca que rebota en el suelo y desaparece a los 10 segundos, al alcanzar a un enemigo o al desaparecer de la pantalla. \* Diseño artístico y audio: Este juego está inspirado en Mario Bros, por lo que usa sprites y música retro (8-16 bits).

b091t0

## Herramientas utilizadas

* Kanban Tasker: Permite organizar el trabajo mediante metodologías ágiles. Funciona agregando tarjetas para cada tarea que se debe realizar y ubicándolas en una columna según su estado (en reserva, por hacer, en proceso, revisar y terminado) y según su prioridad (alta, normal y baja). De esta forma se puede ver el estado del proyecto de forma sencilla.

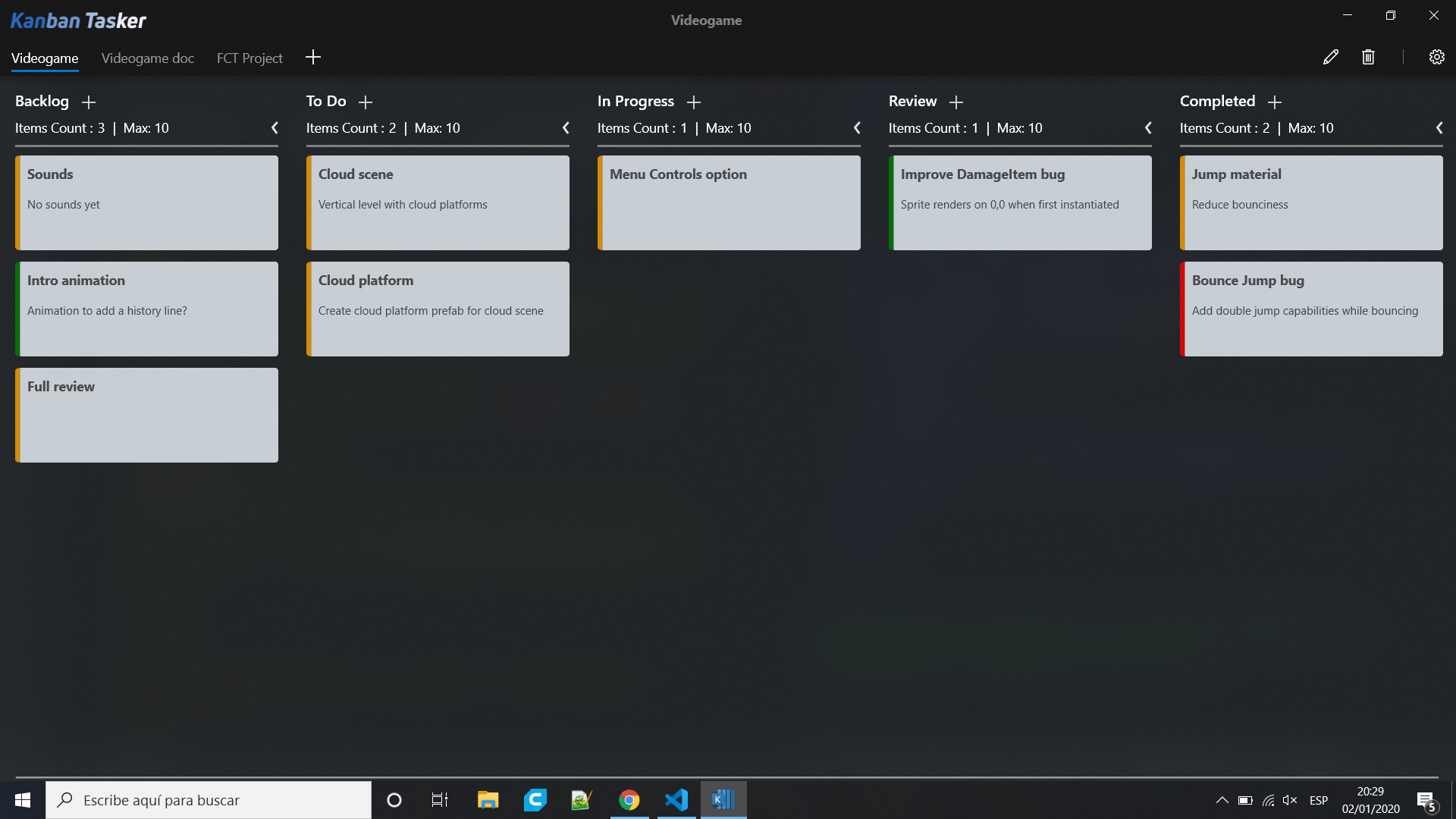


Imagen Kanban Tasker

* Gimp: Editor gratuito de imágenes utilizado para retocar y crear sprites.



Imagen de Gimp

* Visual Studio Code: Editor de texto compatible con Markdown y Mermaid utilizado para escribir esta memoria y exportarla a PDF a través de extensiones:
  + Markdown All in One: Atajos de techado para Markdown.
  + Markdown PDF: Permite exportar los documentos a PDF, HTML, PNG y JPEG.
  + Markdown Preview Mermaid Support: Añade soporte para Mermaid

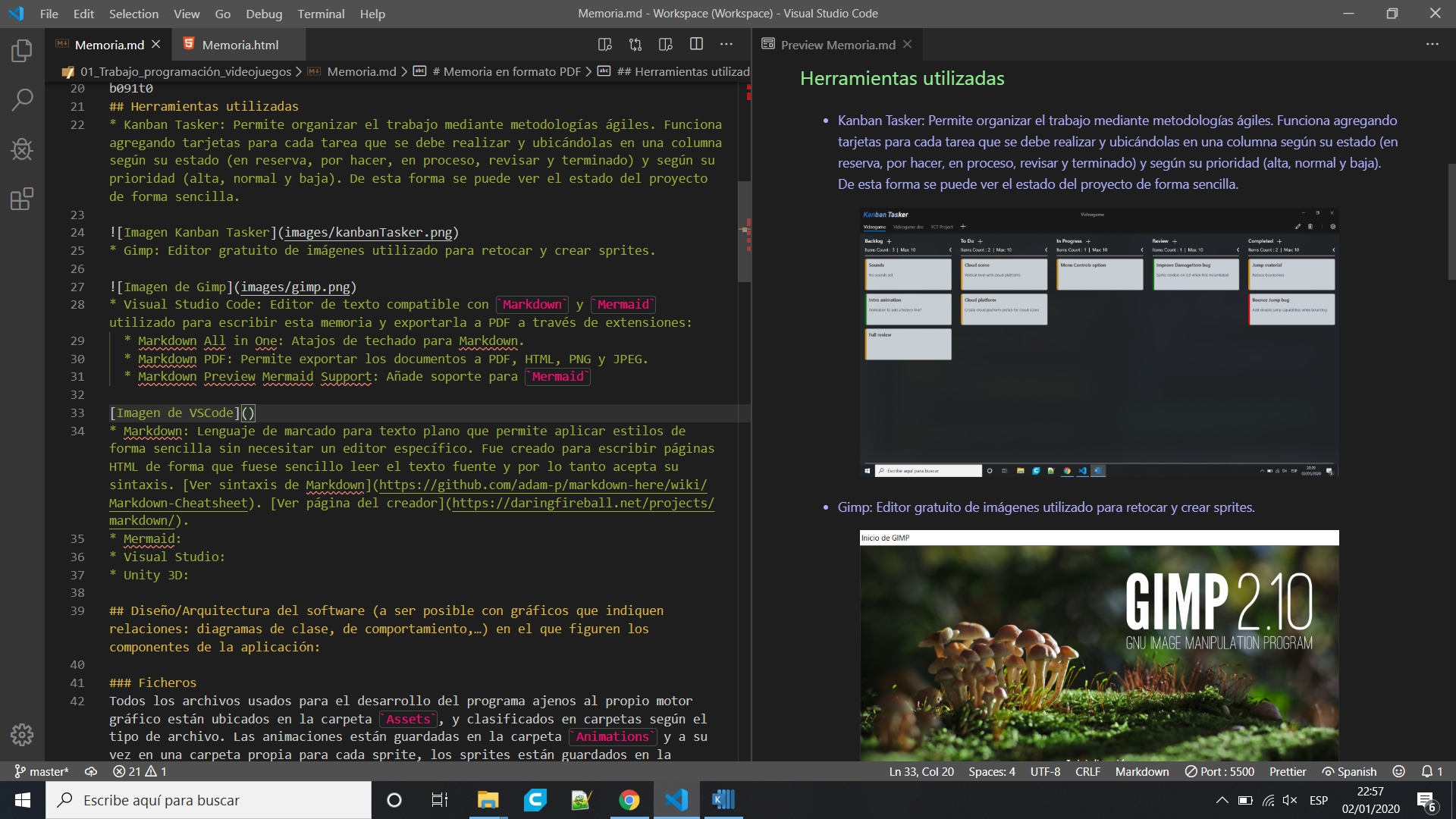


Imagen de VSCode

* Markdown: Lenguaje de marcado para texto plano que permite aplicar estilos de forma sencilla sin necesitar un editor específico. Fue creado para escribir páginas HTML de forma que fuese sencillo leer el texto fuente y por lo tanto acepta su sintaxis. [Ver sintaxis de Markdown](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet). [Ver página del creador](https://daringfireball.net/projects/markdown/).
* Mermaid: Es un script de JavaScript que permite dibujar diagramas a usando la sintaxis de markdown. Permite generar diagramas de flujo, de secuencia, de Gantt, de clases, de estado, de tarta y mas. [Página oficial](https://mermaid-js.github.io/mermaid/#/).
* Mermaid Live Editor: Editor online de Mermaid usado para exportar los diagramas. [Abrir en el navegador](https://mermaidjs.github.io/mermaid-live-editor/).

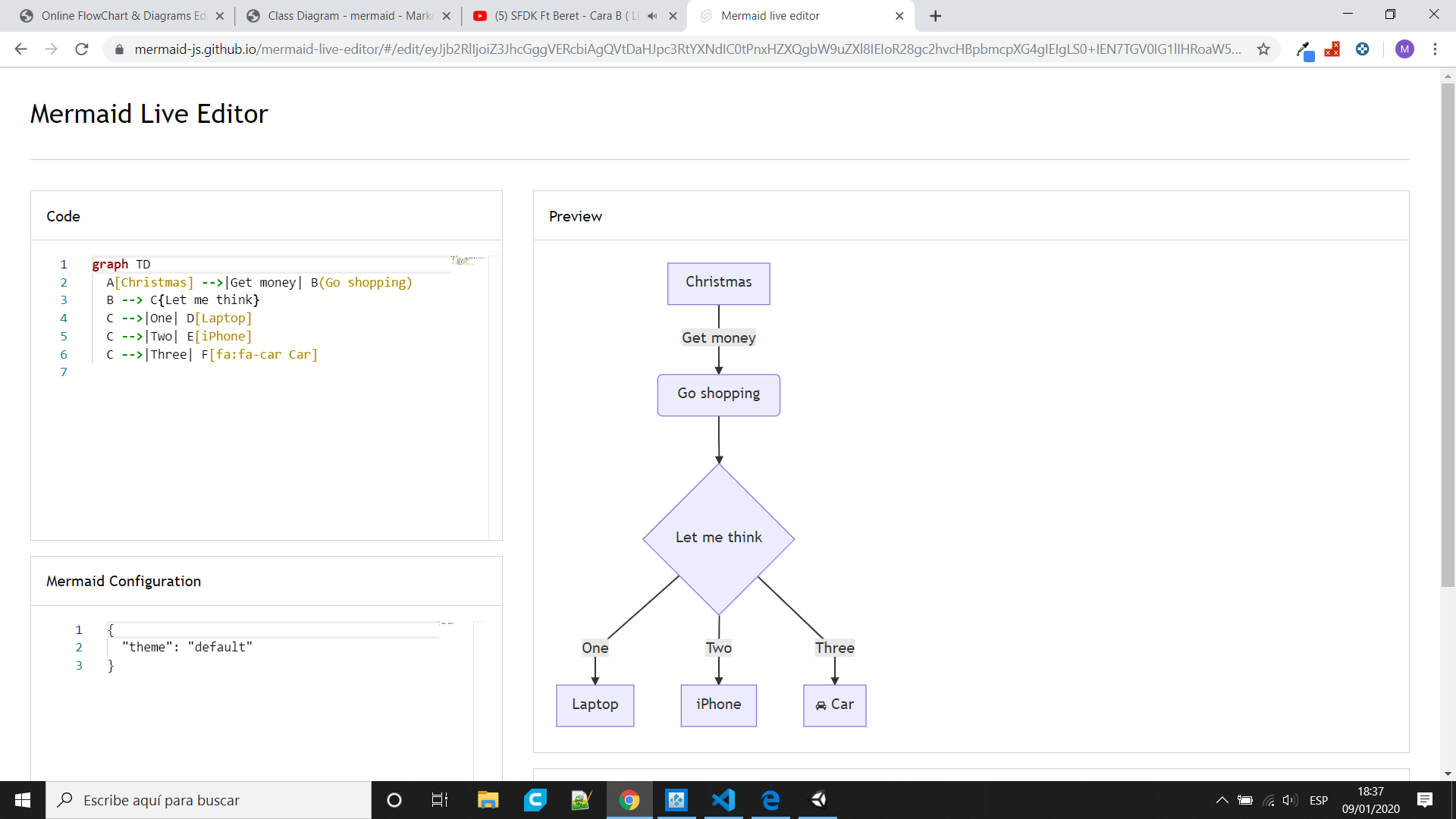


Imagen Mermaid Live Editor

* Visual Studio: IDE compatible con Unity para escribir scripts.

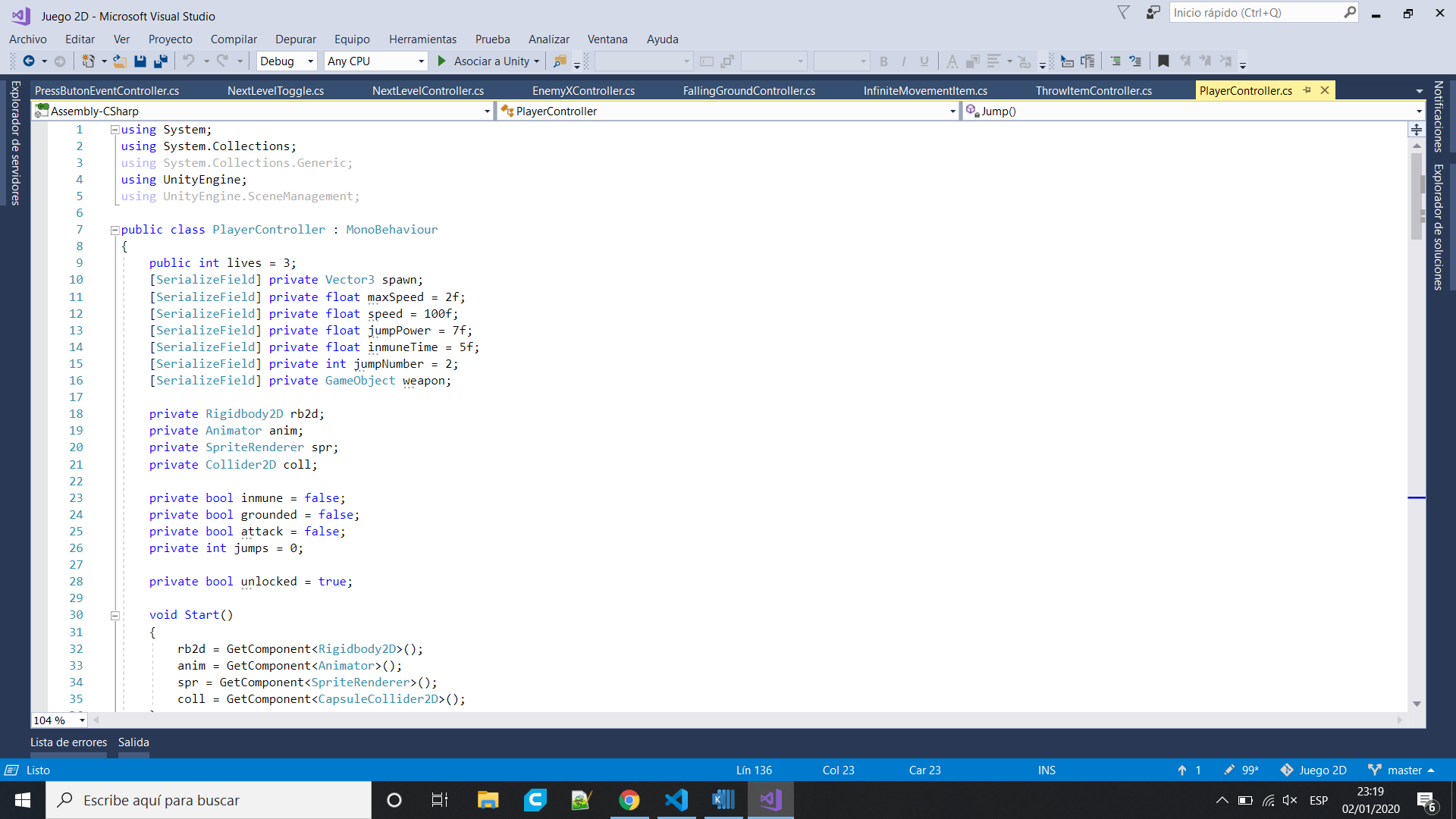


Imagen Visual Studio

* Unity 3D: Motor gráfico.

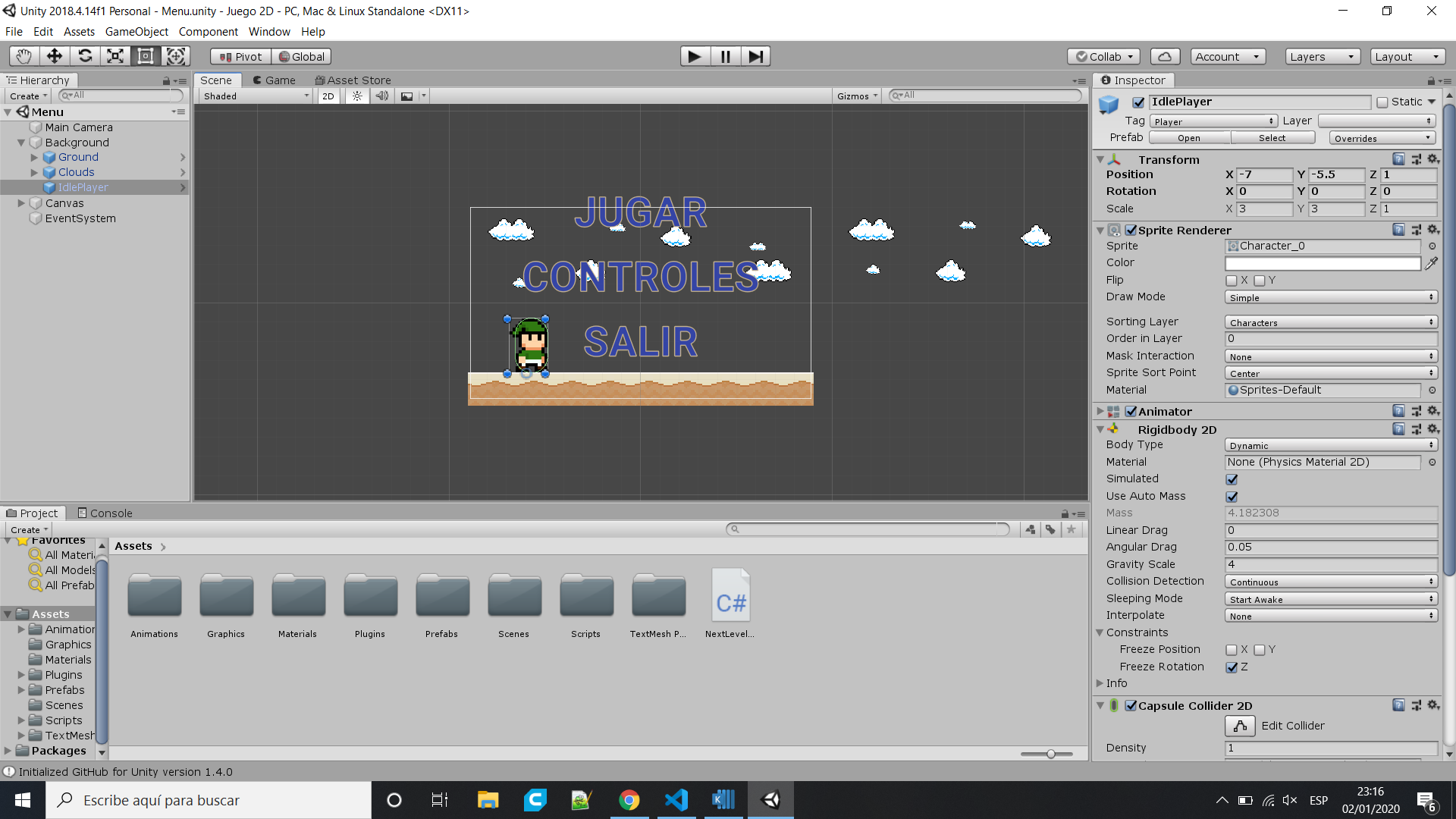


Imagen Unity

## Diseño/Arquitectura del software (a ser posible con gráficos que indiquen relaciones: diagramas de clase, de comportamiento,…) en el que figuren los componentes de la aplicación:

### Ficheros

Todos los archivos usados para el desarrollo del programa ajenos al propio motor gráfico están ubicados en la carpeta Assets, y clasificados en carpetas según el tipo de archivo. La música esta en la carpeta Audio, las animaciones están guardadas en la carpeta Animations y a su vez en una carpeta propia para cada sprite, los sprites están guardados en la carpeta Graphics, los archivos de propiedades como materiales del motor de físicas o propiedades de otros elementos en la carpeta Materials, los objetos creados en el entorno de desarrollo en la carpeta Prefabs, cada uno de los niveles en Scenes y por último los scripts en la carpeta Scripts, ordenados a su vez en carpetas según qué tipo de objetos los utilizan.

├───Animations  
│ ├───Bat  
│ ├───Character  
│ ├───Ghost  
│ └───Spider  
├───Audio  
├───Graphics  
├───Materials  
├───Prefabs  
│ ├───Background  
│ ├───Characters  
│ ├───Objects  
│ └───Platforms  
├───Scenes  
└───Scripts  
 ├───Cameras  
 ├───Characters  
 ├───Logic  
 ├───Menus  
 └───Physics

#### Animaciones

Para animar los sprites son necesarios como mínimo dos archivos: archivos de animación (.anim) y un archivo que controle como se comportan las diferentes animaciones (.controller).

Para crear una animación abrimos la ventana Animation y seleccionamos un objeto de tipo sprite de la escena que tengamos abierta. Arrastramos cada uno de los frames de la animación desde la vista de archivos y ajustamos la línea de tiempo para determinar cuanto será visible cada fotograma. Guardamos en un archivo y repetimos para cada animación.

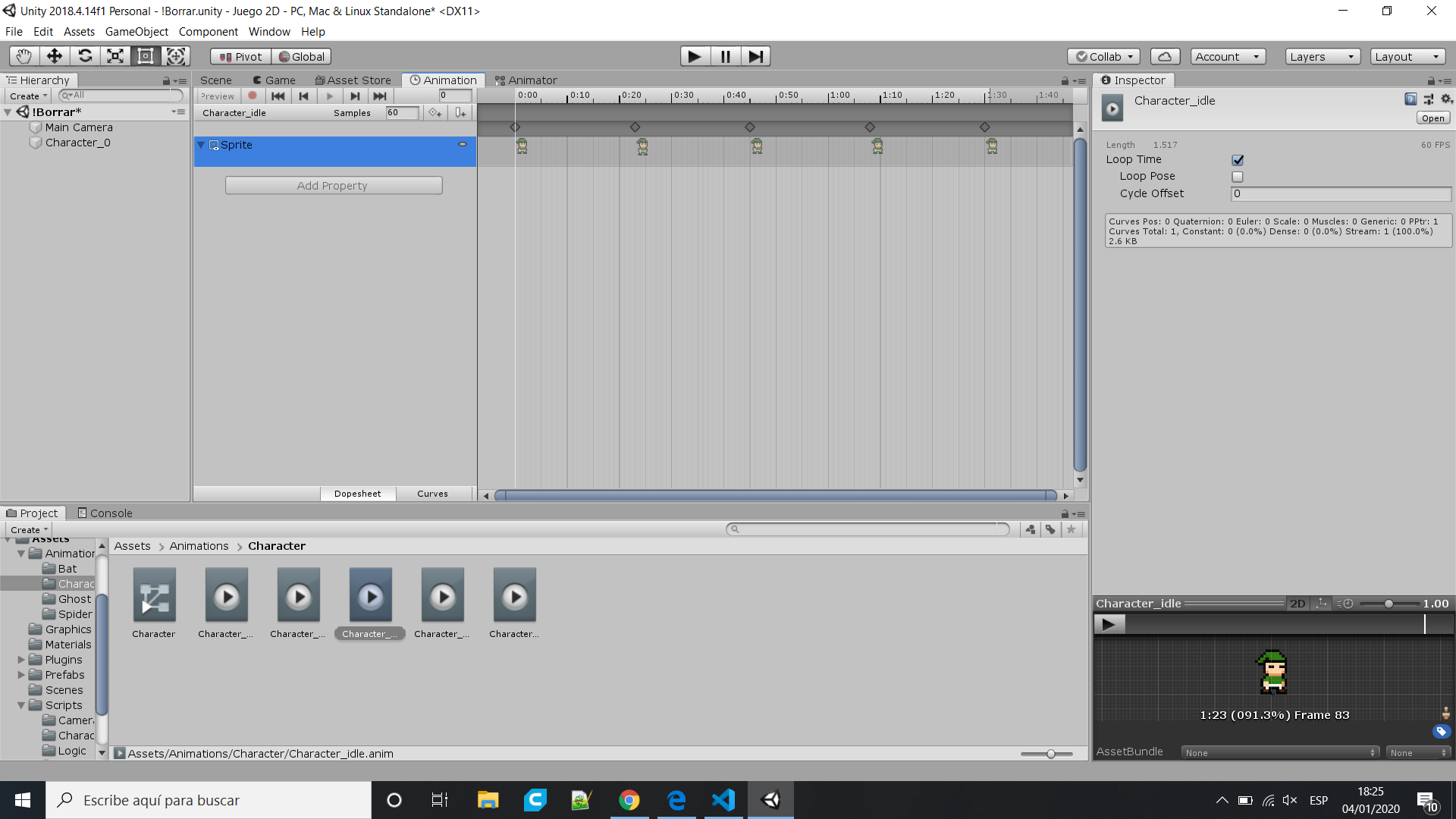


Imagen de Animation

Para mostrar las animaciones en la escena creamos un controlador desde la ventana Animator. Arrastramos las animaciones creadas en el paso anterior y desde la pestaña parameters declaramos las variables que mas adelante utilizaremos para decidir qué animación se debe de mostrar. Sobre cada una de las animaciones hacemos click derecho y seleccionamos ‘Make Transition’. En la ventana Inspector declararemos las condiciones necesarias que se deben cumplir para cambiar de una animación a otra añadiéndolas desde el botón +. Elegiremos la variable y el valor que debe de tener. Los valores de las variables se modifican desde código puesto que son parte del objeto y el controlador se limita a leerlas. Creamos una transición en cada sentido para cada animación y guardamos el archivo.

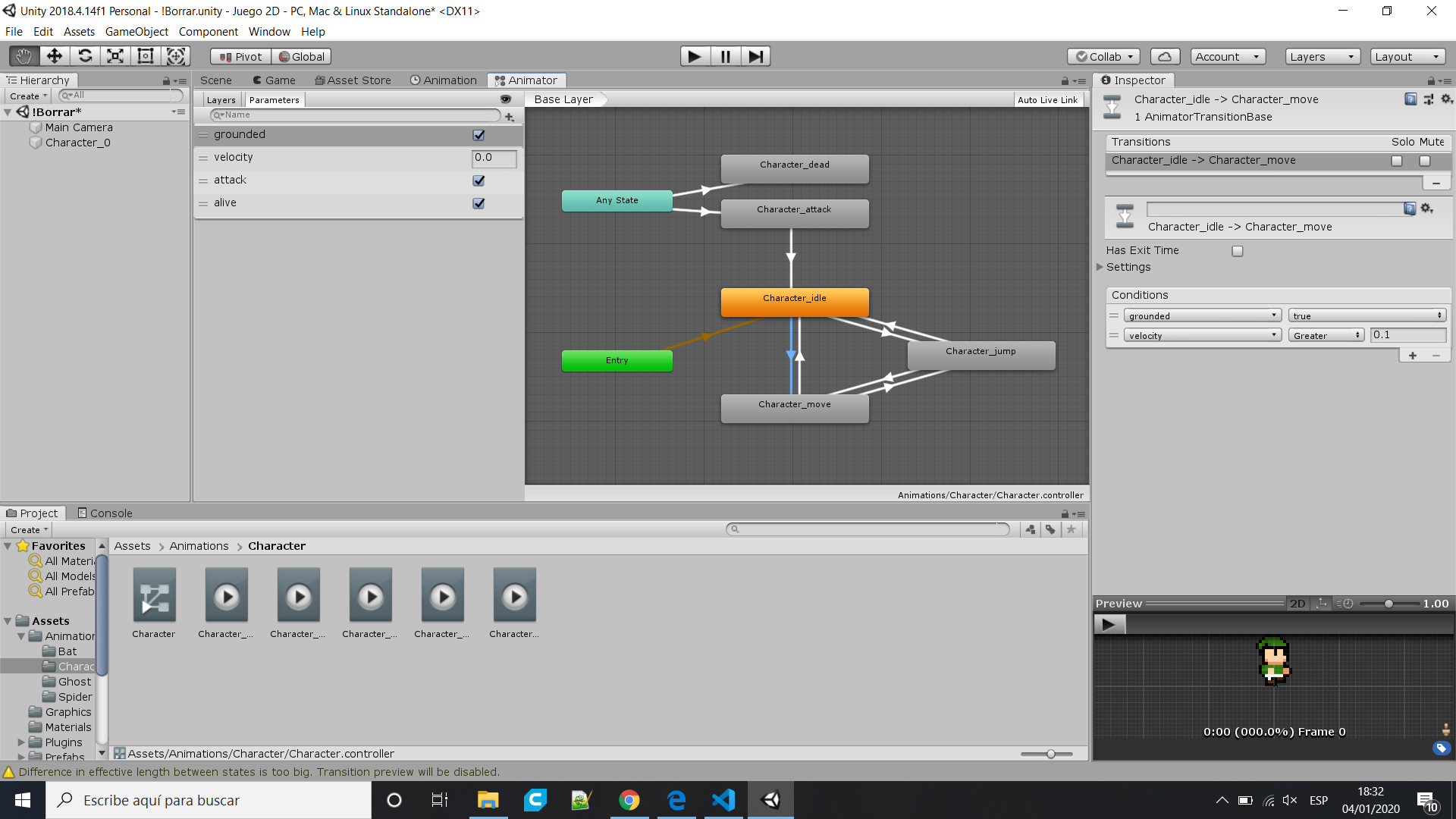


Imagen de Animator

Para cambiar desde código las variables declaradas debemos de añadir un script al objeto en el que se controlará el valor de cada variable. Obtendremos el componente animator al comienzo de la ejecución y asignaremos el valor que queramos a cada variable.

private Animator anim;  
  
void Start()  
{  
 anim = GetComponent<Animator>();  
}  
  
void Update()  
{  
 anim.SetFloat("velocity", 3);  
 anim.SetBool("grounded", true);  
 anim.SetBool("attack", false);  
 anim.SetBool("alive", true);  
}

#### Gráficos (Sprites)

Si la imagen que vamos a utilizar solo contiene un sprite bastará con arrastrar a la escena el archivo, si contiene varios sprites deberemos separarlos desde el editor.

Seleccionamos la imagen y cambiamos el ‘Sprite Mode’ a ‘Multiple’ en la pestaña Inspector. Hacemos click sobre el botón ‘Sprite Editor’ para abrir el editor y buscamos ‘Slice’ en la esquina superior izquierda. En la ventana que se abre pulsamos sobre ‘Slice’ y cerramos el editor. Ahora en la vista de archivos podemos ver que se permiten seleccionar como imágenes diferentes cada uno de los sprites.

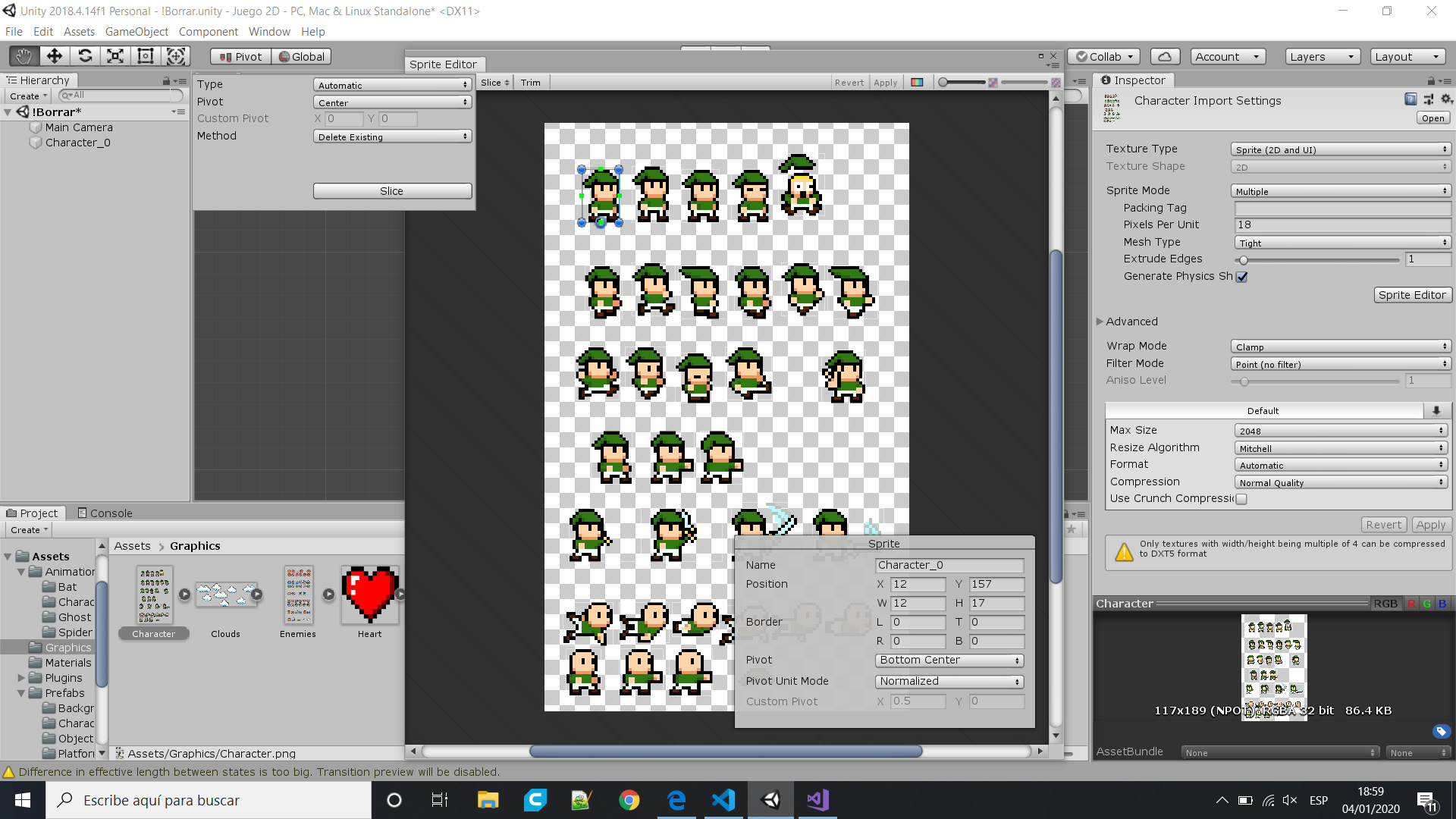


Imagen de Sprites

#### Materiales

Los objetos que trabajan con físicas pueden llevar un PhysicMaterial2D que determina como se comportará al colisionar con otros objetos. Solo contiene dos valores que definen la fricción entre los objetos y el “rebote”.

Para crear un material hacemos click derecho sobre la vista de archivos –> Create –> Phisycs Material 2D.

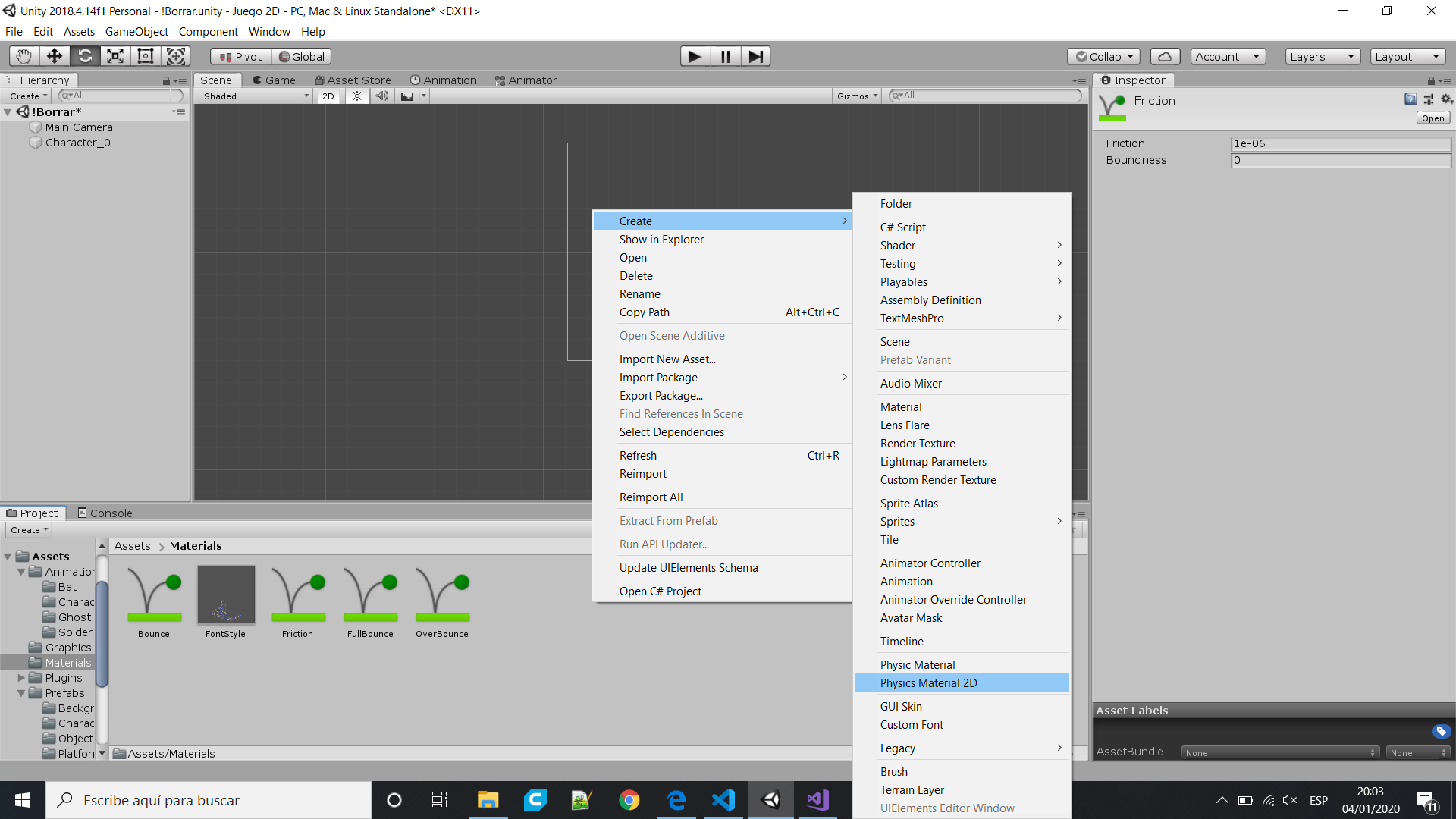


Imagen de material

#### Escenas

Las escenas son cada uno de los diferentes niveles del videojuego. Contienen los objetos, a los que se les define una posición y diversos componentes.

Para crear una escena hacemos click derecho sobre la vista de archivos –> Create –> Scene.

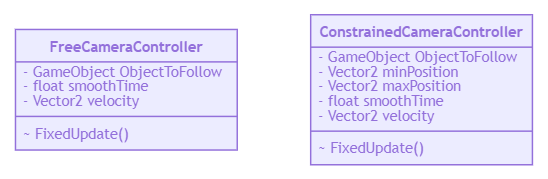
Para cambiar desde código la escena en la que nos encontramos usaremos la clase SceneManager.

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

### Módulos y clases

#### Controlador de la cámara

Estos scripts obtienen la posición del objeto introducido desde la ventana Inspector y desplazan la cámara a esa posición suavizando el movimiento según el parámetro smoothTime. En el caso de la clase ConstrainedCameraController se definen las coordenadas máximas y mínimas del cuadrado en el que se podrá desplazar la cámara, situando la misma en el límite si el objeto a escapa de esa zona. La coordenada mínima debe corresponderse con la esquina inferior izquierda y la máxima con la esquina superior derecha.



CameraClass

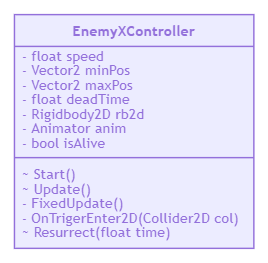
#### Controlador del personaje



CharacterClass

#### Controlador de enemigos

Este script mueve al enemigo al que se le aplica de derecha a izquierda en un área predefinida, cambiando de sentido cuando se alcanza un límite. Solo actúa sobre el eje horizontal. Además “mata” al enemigo cuando detecta una colisión con un objeto etiquetado como DamageItem y lo resucita tras el tiempo especificado por deadTime (en segundos). Definimos el rango en el que se mueve con maxPos y minPos y controlamos su velocidad con speed.



EnemyClass

#### Controlador de la puerta

La puerta puede funcionar de dos maneras distintas, utilizando el prefab NextLevelDoor (puerta desbloqueada) y el Script NextLevelDoor o utilizando todos los demás scripts y el prefab NextLevelButtonDoor.

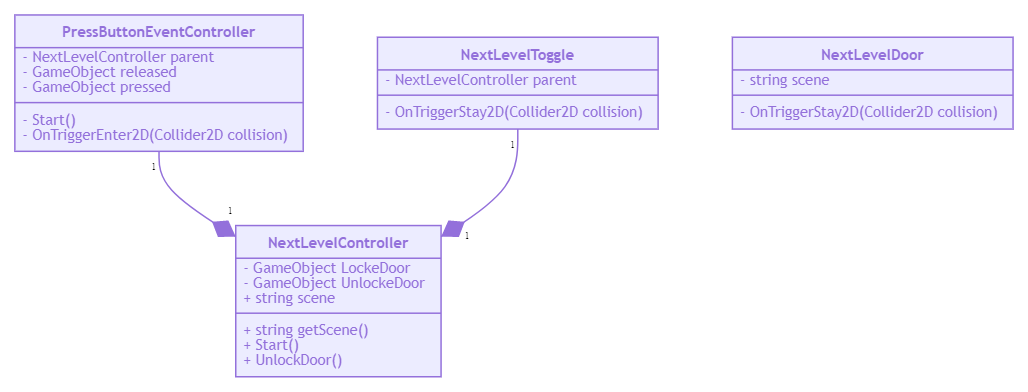
La puerta desbloqueada captura la tecla W y Flecha arriba y cambia la escena a indicada desde el parámetro ‘scene’ cuando se presiona una de las dos cuando el personaje está en contacto con la puerta.

La puerta bloqueada es mas complicada. Para funcionar debe de tener un objeto padre que contenga en su interior el botón, la puerta bloqueada y la puerta desbloqueada. El botón deberá contener en su interior a su vez un botón presionado y otro sin presionar. El elemento padre debe contener el script NextLevelController, la puerta desbloqueada NextLevelToggle y el botón PressButtonEventController. La puerta desbloqueada deberá de estar desactivada.

El script NextLevelController recibe como parámetros los dos objetos de la puerta y la escena a la que se cambiará, y únicamente activará la puerta desbloqueada y desactivará la bloqueada cuando reciba una llamada al método UnlockDoor().

El script NextLevelToggle actúa como NextLevelDoor pero toma como parámetro el script NextLevelContrller del padre y toma la escena indicada ahí.

El script PressButtonEventController recoge como parámetro los botones presionado y liberado, estableciendo como activo el botón liberado y desactivado el botón presionado, invirtiendo los valores cuando se detecta una colisión y llamando al método UnlockDoor() del script MextLevelController del padre recogido como parámetro.



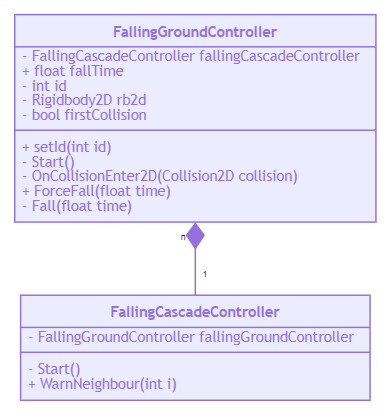
DoorClass

#### Controlador de la plataforma que pierde bloques

Este objeto esta compuesto por varios sprites independientes, con su propio collider y rigidbody. Cada uno de esos sprites aplica el script FallingGroundontroller y el padre utiliza FallingCascadeController.

El script FallingGroundController detecta una colisión y permite a la gravedad actuar en el elemento al que se aplica, y llama al método WarnNeighbour() del script FallingCascadeController introducido por parámetro.

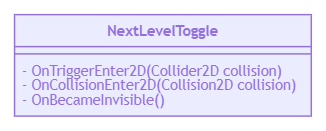
El script FallingCascadeController toma como parámetros los scripts de los sprites independientes y, cuando un script llama a su método WarnNeighbour(), fuerza la caída de los sprites colindantes.



FallingGroundClass

#### Elementos que causan daño (roca)

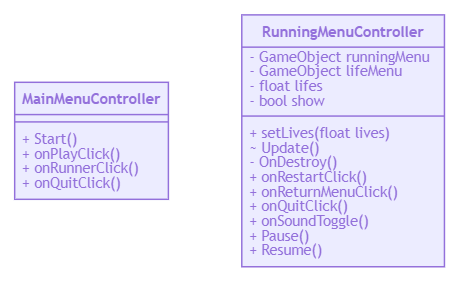
Este script no toma parámetros y se limita a destruir al objeto que lo contiene cuando detecta una colisión contra un enemigo (vivo o muerto) o contra el personaje, o cuando deja de verse en pantalla.



ThrowClass

#### Menús

Los scripts de los menús contienen métodos que utilizarán los botones del menú para realizar acciones.



MenusClass

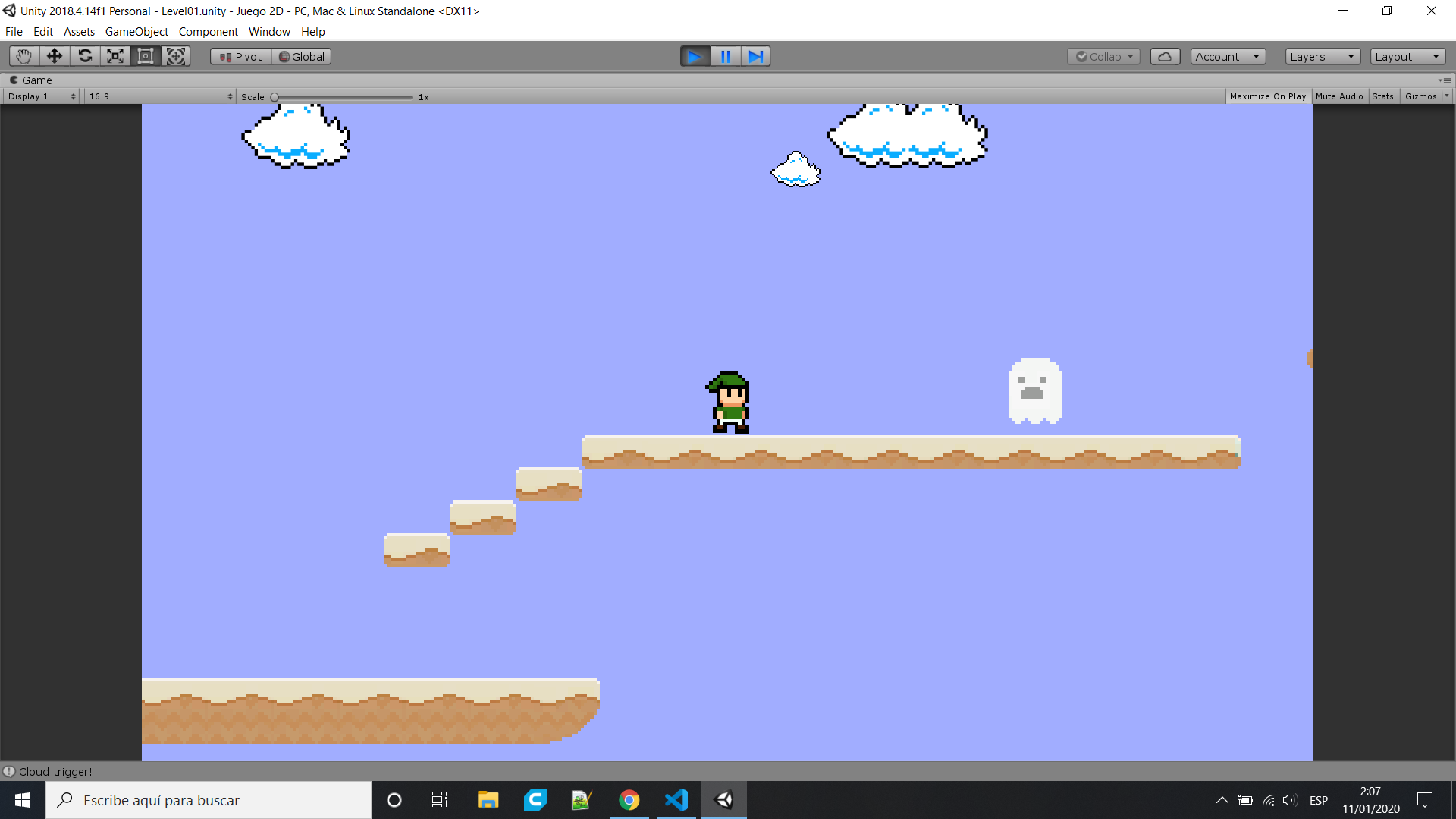
### Escenas

El juego se basa únicamente en un menú y dos niveles puesto que esta sin terminar. Además de esos niveles existe una escena adicional utilizada para probar los diferentes componentes que se van creando durante el desarrollo y que el usuario final no verá.

El menú consiste en un sprite animado del personaje sobre una plataforma, con nubes moviéndose detrás de éste y un menú a su lado. El menú contiene las opciones de “Jugar”, “Runner” y “Salir”.

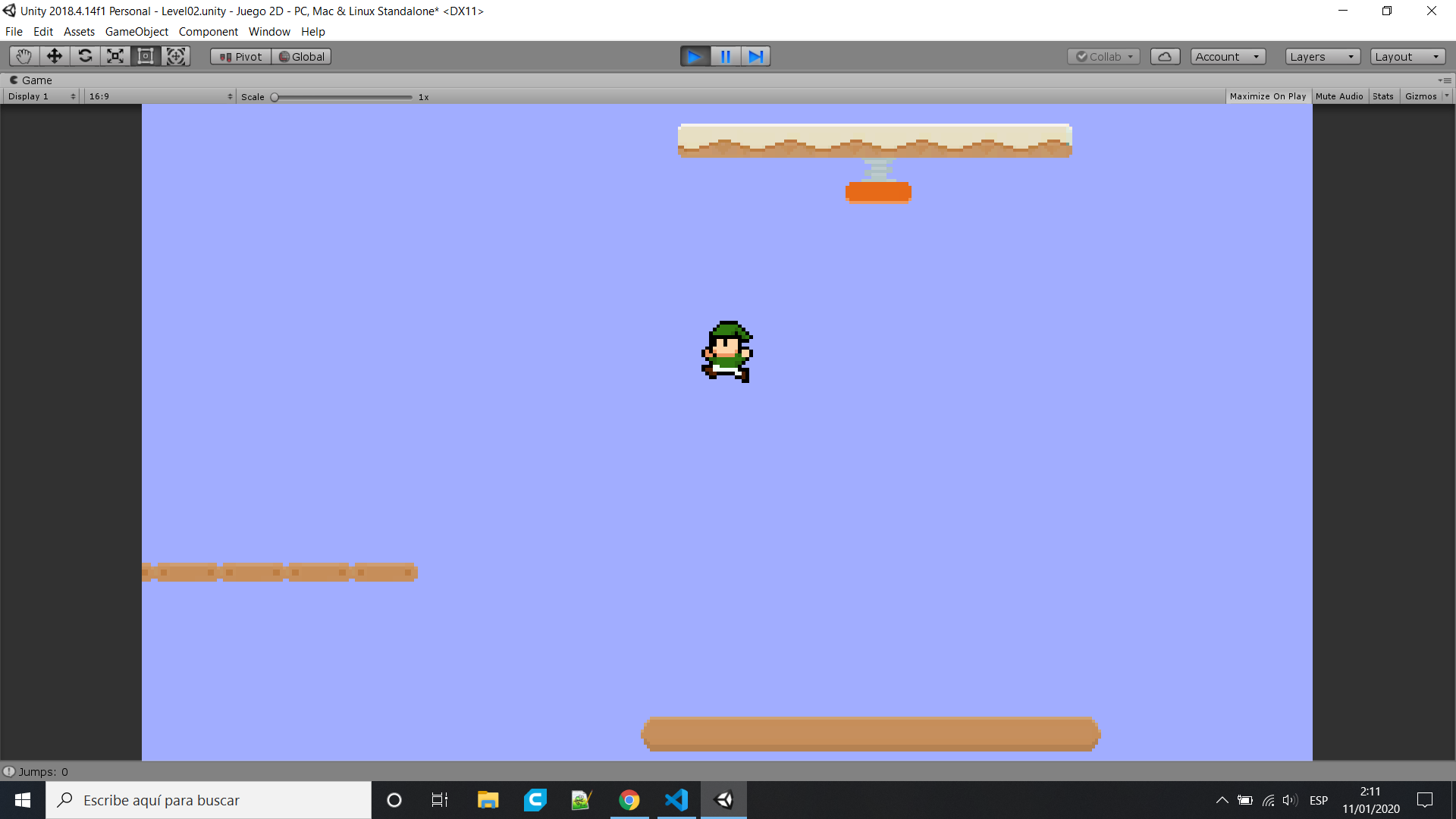
)

El primer nivel consiste en plataformas que conducen al personaje a la puerta que lo llevará al siguiente nivel. Hay una plataforma trampa que no contiene colisionador (por lo que es atravesable), otras que actúan como una cama elástica y por último enemigos “custodiando” la puerta.



Level01Scene

El segundo nivel consiste en unas plataformas que actúan como una cama elástica entre las que hay que avanzar verticalmente hasta alcanzar un botón que desbloquea la última puerta del videojuego y conduce de vuelta al menú.



Level02Scene

### Actores

### Recursos:

En la documentación se pueden encontrar los links a los diferentes sprites utilizados. A excepción de el proyectil (una roca) y la interfaz del menú de pausa todos han sido obtenidos de internet a pesar de que hayan sido modificados levemente.

## Planificación

La idea era empezar con mucho tiempo (noviembre) y trabajar en fragmentos del proyecto cuando me apeteciese, empezar por familiarizarme con Unity y su API y entonces desarrollar las funcionalidades básicas (movimiento del personaje y nivel demo) e ir añadiendo funcionalidades independientes según fuese avanzando el tiempo (crear objetos, plataformas y scripts para guardarlos como ‘prefabs’ y utilizarlos mas tarde) hasta finalmente utilizarlos en diferentes niveles, reservando la última semana para retoques y la documentación. En el hipotético caso de terminar con mucho tiempo de sobra añadiría una modalidad de juego adicional, convirtiendo este juego de plataformas en un ‘runner’.

## Temporización real

07/11/2019 - 14/11/2019:

1. Crear una escena “Demo” en la que trabajar (Demo.scene)
2. Dividir el sprite de las plataformas (Sprites.png) y crear una plataforma con colliders (Suelo09x1.prefab)
3. Añadir animaciones para el personaje y para los enemigos (Spider y Ghost, Bat nunca se llegó a utilizar)
4. Crear el personaje principal añadiendo el sprite (Character.png) y un script para controlar el movimiento (PlayerController.cs, sin saltos ni ataque)
5. Añadir un script a la cámara para que siguiese al personaje. (ConstrainedCameraController.cs y FreeCameraController.cs)
6. Añadir salto al personaje
7. Crear el script para controlar a los enemigos (EnemyXController.cs) y guardar el prefab terminado (Spider.prefab y Ghost.prefab)

14/12/2019 - 21/12/2019:

1. Crear un menú de inicio (Menu.scene)
2. Añadir al menú un personaje animado en una plataforma (IdlePlayer.prefab, PlayerIdleController.cs)
3. Añadir al menú un fondo de nubes en movimiento (MovingClouds.prefab, InfiniteMovementController.cs)
4. Crear plataforma a la que se le caen bloques (FallingGroundController.cs)
5. Crear script adicional para la plataforma anterior que causa que los bloques colindantes se caigan en cascada (FallingCascadeController.cs)

17/12/2019 - 21/12/2019:

1. Añadir puerta para cambiar de nivel (NextLevelDoor.prefab, NextLevelDoor.cs)
2. Cerrar la puerta y añadir botón para abrirla (NextLevelButtonDoor.prefab, NextLevelController.cs, PressButtonEventController.cs)
3. Añadir ataque al personaje (Rock.png)
4. Añadir menú de pausa para la tecla ESC (RunningMenu.prefab, RunningMenuController.cs, Menu.png)

21/12/2019 - 28/12/2019:

1. Guardar las plataformas utilizadas como prefabs.
2. Añadir niveles

04/01/2020 - 14/01/2020:

1. Revisión, optimización y comentado de código
2. Documentación

## Valoración de los resultados

No estoy del todo contento con el resultado. El primer día debí de planificar cuales eran los objetos que iban a formar parte del proyecto, diseñar las clases y dibujar los diagramas para tener claro como debía de hacerlo. En mi defensa diré que al no conocer la API de Unity y no estar familiarizado de ella era prácticamente imposible. Estaba tan perdido que no sabía ni si era posible comunicar los diferentes objetos de la escena entre sí, cosa que ahora veo evidente.

Respecto a la parte del ‘núcleo’ del videojuego (movimiento del personaje y nivel demo) estoy bastante contento, apenas tardé una semana en llevarlos a cabo y funcionan perfectamente (excepto el salto).

El mayor error que cometí en el desarrollo del videojuego, además de la falta de planificación, fue no percatarme de los verdaderos elementos necesarios en cualquier videojuego y que no abordé hasta el final o no tuve tiempo para realizar (menú de pausa, menú principal, audio, sistema de puntuaciones). Personalmente preferiría un proyecto con todo lo básico programado aunque no tuviese niveles propiamente dichos.

Otro de los grandes defectos que veo es un código desorganizado. Si pudiese repetir el proyecto extraería en métodos muchas de las funcionalidades, sobre todo en el script que controla al personaje, y hubiera utilizado los métodos del ciclo de vida para llamar a mis propios métodos en lugar de programar en su interior. A día de hoy (11/01/2020) voy a intentar corregir estos fallos y otros que no he nombrado para entregar el proyecto como considero que deberia ser.

A pesar de todos los defectos ya nombrados considero que al no haber usado assets ni extensiones es comprensible no haber podido completar todo a tiempo.

Resumiendo, debo mejorar la planificación.

## Defectos

* Salto: En algunas ocasiones hay el salto falla y no responde a la tecla la primera vez que se pulsa
* Darse de cabeza con una plataforma al saltar: Al detectar la colisión con un objeto etiquetado como ‘grounded’ la animación detecta que esta encima de la plataforma y cambia la animación de salto.
* Saltos en plataforma elástica: El salto funciona estableciendo la velocidad del eje vertical a 0 y añadiendo una fuerza al personaje cuando se pulsa la tecla de salto, por lo tanto, si el personaje iba a alcanzar una posición mas alta que la que alcanzaría sin saltar, la plataforma elástica, que te hace rebotar hasta la posición vertical mas alta que has alcanzado, olvidará la posición mas alta y la sustituirá por la última, devolviendo al personaje a una posición mas baja de lo que debería.
* Rebotes en plataforma elástica: cuando el personaje esta situado entre dos plataformas elásticas (una encima y otra debajo) y alcanza a rebotar a la superior, esta le devolverá rebotado hacia abajo, y la de abajo hacia arriba, creando un ciclo que aumenta cada vez más la velocidad haciendo incontrolable al personaje.

## Bibliografía/Webgrafía

Implementado: \* [Movimiento del personaje (físicas), escenario y animaciones (vídeos 1-10)](https://www.youtube.com/playlist?list=PLiplYDjUMtti5bWJ1Ugystr-vUg8B5EuP) \* [Crear un terreno o cielo en perpetuo movimiento I](https://www.youtube.com/watch?v=MRThXOxOcuE&t=20s), [II](https://www.youtube.com/watch?v=xAmyzWeEde4) \* [Parar el videojuego](https://answers.unity.com/questions/1230216/a-proper-way-to-pause-a-game.html) \* [Ocultar un elemento al presionar una tecla](https://answers.unity.com/questions/850220/how-can-i-get-a-ui-canvas-to-hideappear-on-esc-but.html) \* [Lanzar proyectiles](https://www.youtube.com/watch?v=zAgAxrTCbGM) \* [Instanciar objetos](https://forum.unity.com/threads/shooting-fireballs.230769/) \* [Destruir objetos](https://docs.unity3d.com/es/530/ScriptReference/Object.Destroy.html) \* [Corrutinas IEnumerator (equivalente a un hilo paralelo)](https://docs.unity3d.com/ScriptReference/WaitUntil.html) \* [Solucionar error: Ejecución empezaba parada](https://answers.unity.com/questions/171656/timetimescalet-to-0-whenever-i-start.html) \* [Cambiar de escena](https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html) \* [Silenciar audio](https://answers.unity.com/questions/52109/how-do-i-mute-all-audio-sound.html)

No implementado: \* [Generar terreno aleatorio](https://www.youtube.com/watch?v=VkGG9Umag0M)

Recursos: \* [Sprite personaje, enemigos y plataformas](https://drive.google.com/file/d/1g6Pytql6h1gFt5WiMpdrV23jvrY2shtU/edit) \* [Sprite nubes](http://www.llumac.cat/picsl/799062/) \* [Sprite corazón](https://opengameart.org/content/heart-pixel-art) \* [Sprite alavoz](https://www.shutterstock.com/es/image-vector/audio-speaker-volume-sound-button-retro-1340123576?id=1340123576&irgwc=1&utm_medium=Affiliate&utm_campaign=Freepik%20Company%2C%20S.L.&utm_source=39422&utm_term=5e19f8cf61855.5e19f8cf61856) \* [Audio](https://www.playonloop.com/2018-music-loops/follow-me/)

Solo se ha usado el antipatrón en los scripts de movimiento del personaje y de la cámara.