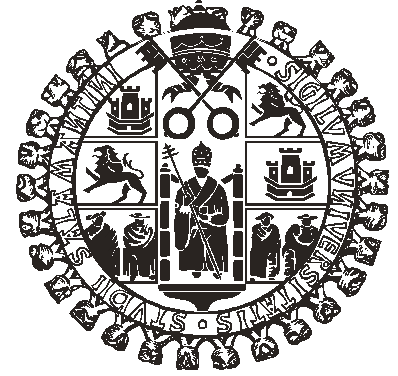
UNIVERSIDAD DE SALAMANCA

ESCUELA POLITÉCNICA SUPERIOR DE ZAMORA

GRADO EN INGENIERÍA INFORMÁTICA EN SISTEMAS DE INFORMACIÓN

**VIDEOJUEGO COOPERATIVO DE PLATAFORMAS EN 2D**



AUTOR: JORGE APELLÁNIZ COLINO

TUTOR: JUAN CARLOS MATOS FRANCO

*Departamento de Informática y Automática*

Fecha de adjudicación: Marzo 2019

Fecha de presentación: Septiembre de 2019

ÍNDICE

[INDICE DE TABLAS 5](#_Toc17131511)

[AGRADECIMIENTOS 6](#_Toc17131512)

[ABREVIATURAS 7](#_Toc17131513)

[1. INTRODUCCIÓN 8](#_Toc17131514)

[1.1. MOTIVACIÓN 8](#_Toc17131515)

[1.2. CONTENIDO DE LA MEMORIA 8](#_Toc17131516)

[1.2.1 DESCRIPCIÓN DEL PROYECTO 9](#_Toc17131517)

[1.2.2 DOCUMENTACIÓN TÉCNICA 9](#_Toc17131518)

[2. OBJETIVOS DEL PROYECTO 9](#_Toc17131519)

[2.1. ALCANCE 11](#_Toc17131520)

[3. CONCEPTOS TEÓRIOS 11](#_Toc17131521)

[3.1. CICLO DE VIDA DE UNITY 11](#_Toc17131522)

[3.1.1. EDITOR 11](#_Toc17131523)

[3.1.2. INICIALIZACIÓN 12](#_Toc17131524)

[3.1.3. UPDATE ORDER (ORDEN DE ACTUALIZACIÓN) 13](#_Toc17131525)

[3.1.4. RENDERING (RENDERIZACIÓN) 14](#_Toc17131526)

[3.1.5. COROUTINES (CORRUTINAS) 15](#_Toc17131527)

[3.1.6. DESTRUCCIÓN DEL OBJETO 16](#_Toc17131528)

[3.1.7. FINALIZACIÓN 16](#_Toc17131529)

[3.2. UML Y LOS DIAGRAMAS DE ESTADOS 16](#_Toc17131530)

[3.2.1. UML 16](#_Toc17131531)

[3.2.2. DIAGRAMAS DE ESTADOS 17](#_Toc17131532)

[4. HERRAMIENTAS 19](#_Toc17131533)

[4.1. MOTOR GRÁFICO UNITY 20](#_Toc17131534)

[4.1.1. ¿QUÉ ES UN MOTOR GRÁFICO? 20](#_Toc17131535)

[4.1.2. ¿QUÉ ES UNITY? 20](#_Toc17131536)

[4.1.3. ¿POR QUÉ UNITY? 21](#_Toc17131537)

[4.1.4. UNITY MULTIPLAYER 21](#_Toc17131538)

[4.1.5. EXPLICACIÓN DE UNITY 24](#_Toc17131539)

[4.2. C# 26](#_Toc17131540)

[4.3. GITHUB 26](#_Toc17131541)

[4.4. FREE PLATAFORM GAME ASSETS 27](#_Toc17131542)

[5. ASPECTOS RELEVANTES DEL DESARROLLO 27](#_Toc17131543)

[5.1. PLANTEAMIENTO DEL PROYECTO 27](#_Toc17131544)

[5.2. METODOLOGÍA DEL CICLO DE VIDA DEL SOFTWARE 28](#_Toc17131545)

[5.2.1. MODELO ITERATIVO INCREMENTAL 29](#_Toc17131546)

[5.3. PLANIFICACIÓN DEL PROYECTO 31](#_Toc17131547)

[5.4. FASES ITERATIVAS 31](#_Toc17131548)

[5.5. IMPLEMENTACIÓN 32](#_Toc17131549)

[5.5.1. DISTRIBUCIÓN DE LOS ASSETS 32](#_Toc17131550)

[5.1.2. SCRIPTS 33](#_Toc17131551)

# INDICE DE TABLAS

# AGRADECIMIENTOS

# ABREVIATURAS

# INTRODUCCIÓN

En este documento se busca reflejar la documentación relativa al Trabajo de Fin de Grado “**Juego cooperativo de plataformas en 2D**” relativo al Grado en Ingeniería Informática en Sistemas de Información de la Escuela Politécnica Superior de Zamora (Universidad de Salamanca).

Mediante el desarrollo y la implementación de esta aplicación informática se busca crear un videojuego, enfocado para ordenadores, basado en los juegos de plataformas, donde la mecánica de movimiento se puede asimilar a juegos conocidos como es *Mario Bros*. Se busca una experiencia de juego rápida y dinámica enfocada a ser disfrutada en pequeños tiempos muertos.

El objetivo es proporcionar un entretenimiento a corto plazo. El público objetivo serían personas con una edad superior a 8 años, A pesar de esto se espera que cualquier tipo de usuario pueda disfrutar de una experiencia de juego satisfactoria.

*Palabras clave: Universidad de Salamanca, Escuela Politécnica Superior de Zamora, Aplicación Informática, Juego cooperativo, Mario Bros, entretenimiento, corto plazo.*

## 1.1. MOTIVACIÓN

Desde principios de los años 80 la industria de los videojuegos ha sufrido un crecimiento enorme, superándose año tras año. Hasta el punto el punto de convertirse en la mayor industria del entretenimiento a nivel mundial, superando a otros mercados del entretenimiento con creces.

Desde pequeño siempre me han gustado los videojuegos y por ese motivo y viendo que puede ser una salida profesional en este momento he decido desarrollar un videojuego con el que no solo sea aprender de esta rama de conocimiento sino también con poder disfrutar jugando a ellos.

## 1.2. CONTENIDO DE LA MEMORIA

La estructura de la memoria principal se ha elegido de acuerdo al modelo de memoria recomendado por la Universidad de Salamanca para la realización de los trabajos de Fin de Grado de la Universidad de Salamanca en: “Proyecto de Final de Carrera en Ingeniería Técnica de Informática: Guía de realización y documentación”, elaborada por los profesores Francisco José García Peñalvo, José Rafael García-Bermejo Giner y María N. Moreno García de la Universidad de Salamanca; Jesús Manuel Maudes Raedo de la Universidad de Burgos; y Mario Gerardo Piattini Velthuis de la Universidad de CastillaLa Mancha También se ha utilizado como referencia un Documento de Diseño de Videojuego (GDD) para el Anexo I.

### 1.2.1 DESCRIPCIÓN DEL PROYECTO

1. Introducción

2. Objetivos del proyecto

3. Conceptos teóricos

4. Técnicas y herramientas

5. Aspectos relevantes del desarrollo y problemas encontrados

6. Trabajos relacionados

7. Conclusiones

### 1.2.2 DOCUMENTACIÓN TÉCNICA

* Anexo I – Documento de diseño de videojuego o GDD. Documento en el que se describe con detalle el videojuego.

# OBJETIVOS DEL PROYECTO

El objetivo principal de este Trabajo Fin de Grado es realizar una aplicación acompañada de una memoria explicativa en la que se pueda ver reflejada la aplicación de los conocimientos teóricos-prácticos adquiridos a lo largo de la etapa formativa, la capacidad de autosuficiencia e innovación mediante el trabajo con nuevas tecnologías.

Para la justificación de estas competencias se realiza un videojuego en 2D con una herramienta llamada “Unity3D”, construyendo la estructura principal de dicha plataforma, los componentes el diseño y las características.

El objetivo principal es la creación de un entretenimiento para un amplio rango de personas y poder satisfacerlas. A nivel personal, el objetivo es adquirir una serie de conocimientos que puedan abrirme puertas al mundo laboral en el desarrollo de videojuegos:

* ***Game Designer****:* Se busca adquirir las competencias necesarias para poder definir la estructura y todos los elementos que interactúan en un videojuego. Ser capaz de buscar un equilibrio en todos los elementos que compongan un videojuego para logar ser lo suficientemente atractivo para el público.
* **Programación:** Adquirir los conocimientos necesarios en lenguajes de programación los cuales se utilizan para el desarrollo de videojuegos, como pueden ser C# o JavaScript. En este caso el desarrollo de la aplicación se ha realizado en C#.
* **Motor Gráfico:** Familiarización con uno de los muchos motores gráficos que existen actualmente en el mercado. Conociendo algunas de sus posibilidades y aprendiendo las diferentes mecánicas de trabajo. Desarrolladas a un nivel sencillo, pero pudiendo ser extrapoladas a grandes proyectos.
* ***Tester:*** Esta parte es la encargada de que antes de que se produzca el lanzamiento de un juego se hayan encontrado todos los errores de programación, traducción, de diseño o de doblaje, que en nuestro proyecto será a muy pequeña escala, pero en las grandes super producciones son tareas realmente arduas y complejas a las cuales se destinan mucho tiempo y dinero. Por mi parte se buscará que la aplicación se presente sin ningún tipo de error que pueda afectar a la experiencia del usuario.

En cuanto a los objetivos asociados a la aplicación se busca:

* **Diseño de óptimo de niveles:** Se buscará que los niveles sean consistentes y funcionen correctamente, proporcionando una experiencia de juego satisfactoria.
* **Interfaz de usuario adecuada:** Que la interfaz sea clara y concisa, la cual sea fácilmente entendible por todo el mundo.
* **Tratamiento de los componentes de sonido que utilizará la aplicación.**
* **Programación de las funcionalidades y la lógica de juego**: Mediante scripts realizados en C#.

Como objetivos a más largo plazo, y a modo de mejorar la aplicación en futuras versiones se buscarán los siguientes objetivos:

* **Subida del videojuego a un servidor:** El videojuego al ser cooperativo está preparado para 2 jugadores en una conexión local. En un futuro un buen objetivo es subirlo a un servidor para que los jugadores puedan jugar online desde dos puntos distintos.
* **Mejora de la inteligencia artificial:** En futuras versiones se le puede aplicar otra inteligencia artificial más avanzada que la que hay implementada en este videojuego (explicación posterior).
* **Inclusión de más niveles:** En futuras versiones se buscará incluir muchos más niveles del juego variando su dificultad para qué mejores jugadores de videojuegos les sea más complicado superarlos.

## 2.1. ALCANCE

El videojuego va dirigido a personas de más de 8 años. Estará enfocado para ser jugado en pequeños tiempos muertos. Por lo tanto, se buscará una acción rápida y dinámica, en la que la diversión se obtenga muy rápidamente, y no sea necesarias muchas horas de juego para poder disfrutar de él.

# CONCEPTOS TEÓRIOS

## CICLO DE VIDA DE UNITY

Los objetos que hay dentro de un proyecto de Unity pueden llevar o no asociados scripts que van a describir el comportamiento de dichos objetos. Para entender cómo son leídos y ejecutados por parte de Unity, es necesario explicar primero cómo funciona el ciclo de vida de Unity. En el scripting de Unity, hay un número de funciones de evento que no son ejecutadas en el orden predeterminado como un script ejecuta. Este orden de ejecución se puede describir así:

### EDITOR

* **Reset**: El reset es llamado para inicializar las propiedades de script cuando es por primera vez adjuntado al objeto y también cuando el comando *Reset* es utilizado.

### INICIALIZACIÓN

Estas funciones son llamadas cuando la escena comienza (una para cada objeto en la escena).

* **Awake**: Esta función siempre se llama antes de cualquier función *Start* y también justo después de que un *prefab* es instanciado. (Si un objeto de juego o *GameObject* está inactivo durante el comienzo, *Awake* no es llamado hasta que se vuelva activo).
* **OnEnable**: (solamente es llamado si el objeto está activo): Esta función es llamada justo después de que el objeto es activado. Esto sucede cuando una instancia del comportamiento base (*Mono Behaviour*) es creada, tal como cuando un nivel es cargado o un *GameObject* con un componente script es instanciado.
* **OnLevelWasLoaded**: Esta función es ejecutada para informarle al juego que un nuevo nivel ha sido cargado.

Hay que tener en cuenta que, para todos los objetos agregados a la escena, las funciones *Awake* y *OnEnable* para todos los scripts serán llamados antes de que *Start*, *Update*, etc sean llamados en cualquiera de ellos. Naturalmente, esto no se puede lograr cuando un objeto es instanciado durante el transcurso del juego.

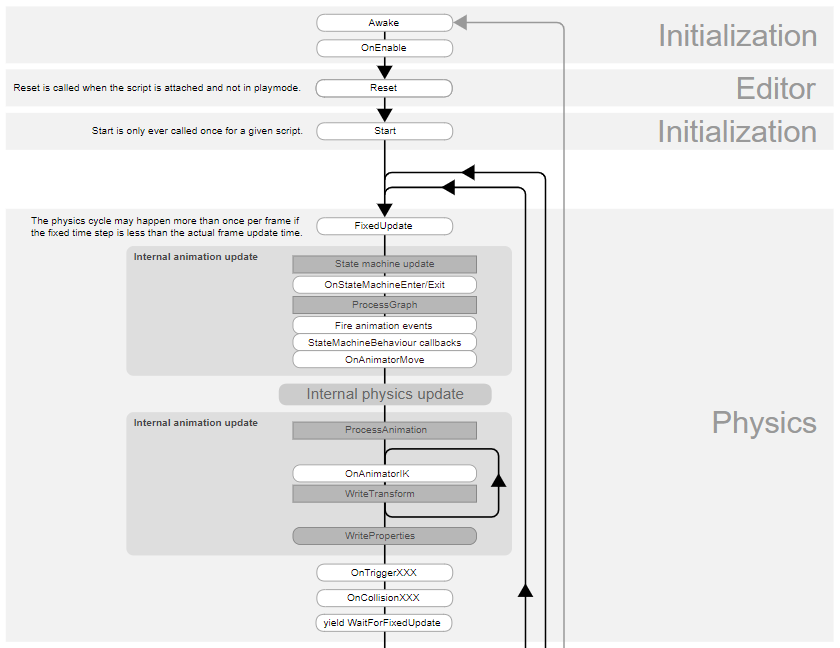


Figura 1. Primer tramo del ciclo de vida

### UPDATE ORDER (ORDEN DE ACTUALIZACIÓN)

Cuando se hace seguimiento de la lógica de juego y las interacciones, animaciones, posiciones de cámara, etc., hay unos eventos diferentes que pueden ser utilizados. El patrón común es realizar la mayoría de tareas dentro de la función *Update*, pero también hay otras funciones que pueden usarse.

* **FixedUpdate**: *FixedUpdate* a veces es más llamada que *Update*. Puede ser llamada varias veces por *frame*, si la velocidad de *frame* es baja y puede no ser llamada entre *frames* en absoluto si la velocidad de *frame* es alta. Todos los cálculos de física y actualizaciones ocurren inmediatamente después de *FixedUpdate*.
* **Update**: *Update* se llama una vez por *frame*. Es la función principal para las actualizaciones de *frames*.
* **LateUpdate**: *LateUpdate* es llamada una vez por *frame*, después de que *Update* haya finalizado. Cualquier cálculo que sea realizado en *Update* será completado cuando *LateUpdate* comience. Un uso común para *LateUpdate* sería una cámara de tercera persona que sigue.

### RENDERING (RENDERIZACIÓN)

* **OnPreCull**: Llamado antes de que la cámara corte (*culls*) la escena. *Culling* determina qué objetos son visibles a la cámara. *OnPreCull* es llamado justo antes de que el *culling* tome lugar.
* **OnBecameVisible/OnBecameInvisible**: Llamado cuando un objeto se vuelve visible/invisible a cualquier cámara.
* **OnWillRenderObject**: Se llama una vez por cada cámara si el objeto es visible.
* **OnPreRender**: Llamado antes de que la cámara comience a renderizar la escena.
* **OnRenderObject**: Llamado después de que toda la renderización regular de la escena es hecha.
* **OnPostRender**: Llamado después de que una cámara finalice de renderizar la escena.
* **OnRenderImage**: Llamado después de que la renderización de escena esté completa para permitir un post-procesado de la imagen de la pantalla.
* **OnDrawGizmos**: Utilizado para dibujar *Gizmos* en la vista de escena por propósitos de visualización. Los *Gizmos* son elementos que se utilizan para proporcionar depuración visual o ayudas de configuración en la vista de la escena de juego.

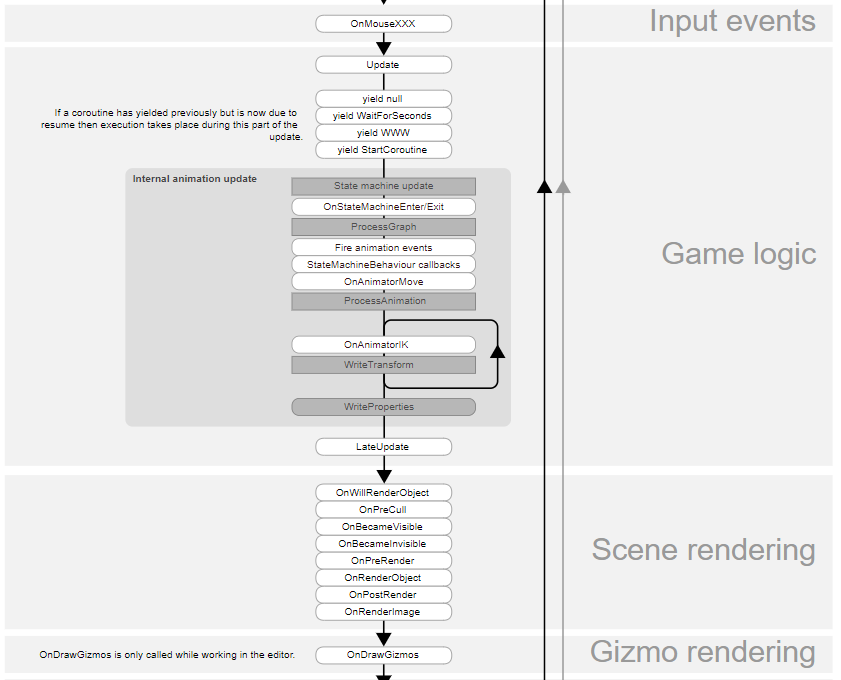


Figura 2. Segundo tramo de ciclo de vida

### COROUTINES (CORRUTINAS)

Las actualizaciones normales de *Coroutines* (corrutinas) son ejecutadas después del valor de retorno que hace la función *Update*. Una corrutina es una función que puede suspender su ejecución hasta que la *YieldInstruction* finalice. Diferentes usos de corrutinas:

* **yield**: La corrutina va a continuar después de que todas las funciones *Update* hayan sido llamadas en el siguiente *frame*.
* **yield** **WaitForSeconds**: Continúa después de un retraso de un tiempo específico, después de que todas las funciones *Update* hayan sido llamadas para el *frame*.
* **yield** **WaitForFixedUpdate**: Continua después de que todos los *FixedUpdate* hayan sido llamadas en todos los scripts.
* **yield** **WWW**: Continúa después de que una descarga WWW haya sido completada.
* **yield** **StartCoroutine**: Se encadena la corrutina, y va a esperar a que la corrutina *MyFunc* haya sido completado primero.

### DESTRUCCIÓN DEL OBJETO

* **OnDestroy**: Esta función es llamada después de que todas las actualizaciones de *frame*, para el último *frame* de la existencia del objeto (el objeto puede ser destruido en respuesta a *Object.Destroy* o al cerrarse la escena).

### FINALIZACIÓN

Sus funciones son llamadas en todos los objetos activos en la escena:

* **OnApplicationQuit**: Esta función es llamada en todos los *GameObjects* antes de que se salga de la aplicación. En el editor es llamada cuando el usuario para el modo de reproducción.
* **OnDisable**: Esta función es llamada cuando el comportamiento se vuelve inactivo o deshabilitado.

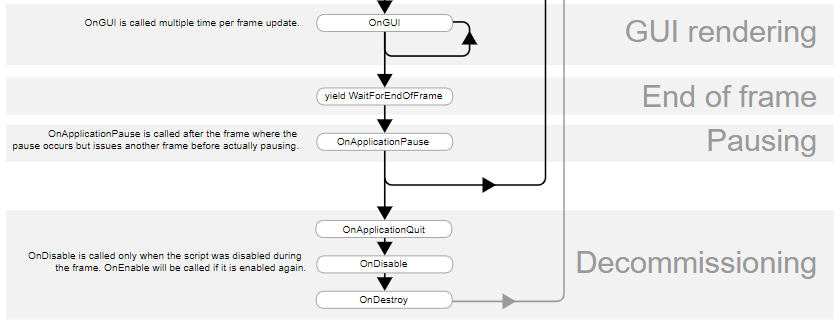


Figura 3. Tercer tramo de ciclo de vida

## UML Y LOS DIAGRAMAS DE ESTADOS

Algunos de los *GameObjects* del proyecto contaban con diversas animaciones para diferentes situaciones. La gestión de todas estas animaciones la he realizado mediante diagramas de estados de UML.

### UML

En un primer momento explicaré brevemente que es UML para poder entender mejor los diagramas de estado que se explican a continuación.

El Lenguaje Unificado de Modelado (UML) fue creado para forjar un lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento. UML tiene aplicaciones más allá del desarrollo de software, p. ej., en el flujo de procesos en la fabricación.

Es comparable a los planos usados en otros campos y consiste en diferentes tipos de diagramas. En general, los diagramas UML describen los límites, la estructura y el comportamiento del sistema y los objetos que contiene.

UML no es un lenguaje de programación, pero existen herramientas que se pueden usar para generar código en diversos lenguajes usando los diagramas UML. UML guarda una relación directa con el análisis y el diseño orientados a objetos.

En los lenguajes orientados a objetos, los algoritmos se expresan definiendo 'objetos' y haciendo que los objetos interactúen entre sí. Esos objetos son cosas que deben ser manipuladas y existen en el mundo real. Pueden ser edificios, artefactos sobre un escritorio o seres humanos.

Los lenguajes orientados a objetos dominan el mundo de la programación porque modelan los objetos del mundo real. UML es una combinación de varias notaciones orientadas a objetos: diseño orientado a objetos, técnica de modelado de objetos e ingeniería de software orientada a objetos [3].

### DIAGRAMAS DE ESTADOS

Un diagrama de estados, en ocasiones conocido como diagrama de máquina de estados, es un tipo de diagrama de comportamiento en el Lenguaje Unificado de Modelado (UML) que muestra transiciones entre diversos objetos.

Ahora bien, ¿qué es un diagrama de estados en UML?

Un diagrama de estados es cualquier dispositivo que almacena el estado de un objeto en un momento dado y puede cambiar el estado o causar otras acciones según la entrada que reciba. Estados se refiere a las diferentes combinaciones de información que un objeto puede mantener, no la forma en que el objeto se comporta. Para comprender los diferentes estados de un objeto, se puede visualizar todos los estados posibles y mostrar cómo un objeto llega a cada de uno de ellos, y eso se puede hacer con un diagrama de estados UML [4].

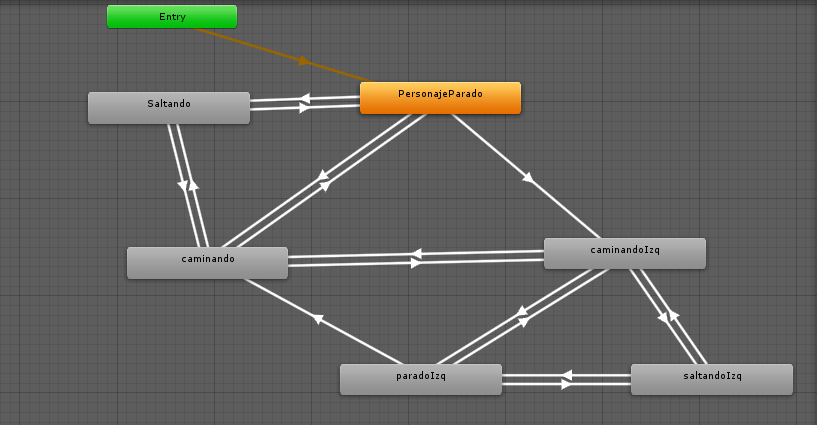


Figura 4. Diagrama de estados de las animaciones del jugador en Unity

En la Figura 4 se muestra el diagrama de estados de las animaciones del jugador. Los estados se representan mediante rectángulos y las transiciones entre los estados con una línea con flecha entre 2 estados. Para cambiar de un estado a otro existen parámetros los cuales si se cumple una serie de condiciones se realiza esa transición.



Figura 5. Parámetros de las animaciones

Por ejemplo, si analizamos la transición siguiente:

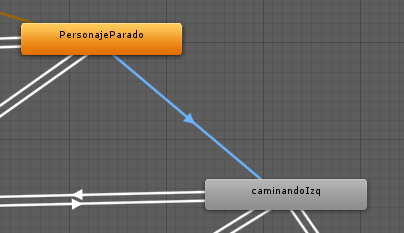


Figura 6. Transición entre personaje parado y caminando hacia la izquierda

La transición que se muestra en la Figura 6 es el cambio de animación de cuando el personaje está parado a cuando se mueve caminando hacia la izquierda. Esa transición se realiza cuando se cumple lo siguiente:

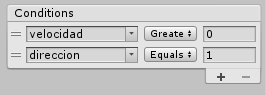


Figura 7. Condiciones de la transición

Cuando la velocidad es mayor que 0 (es decir el personaje se está moviendo) y cuando la dirección es igual a 1, estipulando que el movimiento hacia la izquierda tiene el valor de la dirección en 1 y el movimiento hacia la derecha el valor 0.

Para poder acceder a esos valores mediante un Script podemos ver en la imagen siguiente como se realiza:

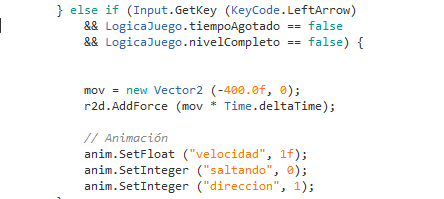


Figura 8. Fragmento de código correspondiente a dicho ejemplo

En este fragmento se expone que cuando se pulsa la tecla de dirección izquierda el jugador se mueve hacia la izquierda y se asignan los valores a los parámetros de la animación. Poniendo la velocidad el valor 1, saltando a 0 (eso es que el jugador no está saltando) y la dirección a 1 (mirando hacia la izquierda).

# HERRAMIENTAS

En este apartado se exponen todas las herramientas que se han usado en el desarrollo del proyecto.

## MOTOR GRÁFICO UNITY

### ¿QUÉ ES UN MOTOR GRÁFICO?

En primer lugar, se debería definir que es un motor gráfico. Son un tipo de software que provee a los desarrolladores y diferentes artistas una herramienta de renderizado. Esto permite plasmar los diseños de los artistas en papel en un modelo 3D, para ello se aplican esquemas 3D a estos dibujos completando finalmente modelos completos.

Los motores gráficos permiten también crear las físicas de los juegos, ofreciendo una gran cantidad de herramientas para ello. En la creación de un juego hay que tener en cuenta el movimiento de los elementos, del personaje, la forma que tienen de interactuar los diferentes objetos con ellos mismos y con los personajes, lo que mediante el uso de las herramientas antes mencionadas se facilita de gran manera.

Los motores gráficos aportan lo necesario para crear las luces y la ambientación de las localizaciones de los videojuegos, facilitan los reflejos y la creación de diferentes materiales.

A fin de cuentas, estos softwares proporcionan a los diferentes profesionales del sector del videojuego un esqueleto sobre el que construir sus ideas y proyectos y facilitando el trabajo para poder centrarse en otros aspectos del desarrollo, como pudieran ser la historia, mecánicas o el diseño de los niveles.

### ¿QUÉ ES UNITY?

Unity es un motor de juego multi-plataforma desarrollado por la empresa *Unity Technologies*. Se anunció y lanzó por primera vez en junio de 2005 en *la Apple Inc.'s Worldwide Developers Conference* como un motor de juego exclusivo para OS X. En 2018 el motor se ha extendido para soportar 27 plataformas diferentes de juego. Este motor se puede utilizar para crear juegos tanto 3D como 2D, además de simulaciones cinematográficas, para ordenadores, portátiles, consolas domésticas, *SmartTVs* y dispositivos móviles. Se han ido lanzado una gran cantidad de versiones desde su lanzamiento, siendo la última versión estable Unity 2019.2.1, lanzada el 14 de agosto de 2019.

Unity proporciona a los usaurios la posibilidad de crear juegos en 3D y 2D, para ello ofrece una API primaria de *scripting* en el lenguaje C#, tanto para el Editor de Unity en forma de *plugins* como para los juegos, así como la funcionalidad de arrastrar y soltar los elementos en la escena (*Drag&Drop*).

Dentro de este programa existen 4 tipos de licencias diferentes, en las que cambian diferentes tipos de funcionalidades. Para el desarrollo de este trabajo se ha utilizado la versión personal gratuita, la cual proporcionaba herramientas suficientes para el fin que se quería conseguir.

* **Personal**: Soporta todas las funciones principales del motor excepto la de D*ark Skin* para la interfaz de usuario en el editor. Además, fuerza a tener una pantalla de inicio (*Splash Screen*) embebida en los juegos.
* **Plus**: Elimina la restricción de *Splash Screen*, aumenta la capacidad de ingreso y el total de usuarios concurrentes al mismo tiempo.
* **Pro**: Garantiza el acceso al suporte Premium, así como informes de rendimiento.
* **Enterprise**: Garantiza el acceso al código fuente del motor, y ofrece una solución multijugador-personalizada.

### ¿POR QUÉ UNITY?

La idea principal del proyecto siempre fue realizar un videojuego de plataformas en 2 dimensiones y posteriormente surgió la idea de realizarlo para multijugador para hacer más llamativo el videojuego. Después de valorar distintas opciones que existen en el mercado se decidió realizar el proyecto un Unity debido el motor reunía las características idóneas para el proyecto. Una gran influencia al elegir este motor gráfico fue el contacto que se tuvo con la herramienta en la asignatura de “Videojuegos” del segundo semestre del 4º curso del grado. Para este proyecto se ha utilizado la versión Unity 2017.3 ya que es con la que estoy más familiarizado por la asignatura.

### UNITY MULTIPLAYER

Al implementar la opción de multijugador, Unity trae una herramienta llamada *UNET* (*Unity* *Multiplayer*), la cuál hizo posible dicha implementación.

*Unity* *Multiplayer* es una de las mejores formas para crear juegos multijugador de forma rápida y sencilla. Dicho proyecto está preparado para que los jugadores se conecten mediante una conexión local. Pero Unity proporciona una serie de servidores (*Unity Matchmaker Servers*) que garantizan que los jugadores puedan encontrarse y jugar entre sí. [UNET].

Esta herramienta ahora mismo se está migrando a otra para su mejora.

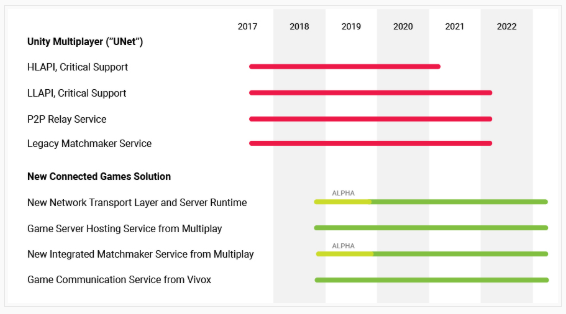


Figura 9. Diagrama de migración de la herramienta UNET

Como se puede observar en la Figura 9, esta herramienta desaparecerá a partir del año 2021.

La elección de realizar el videojuego utilizándola es que cuando se empezó a desarrollar el juego el nuevo sistema de multijugador todavía estaba en las primeras fases de desarrollo y podía contener muchos fallos.

#### FUNCIONAMIENTO DE UNET

La principal característica de esta herramienta es un componente Manager, donde se configuran todos los aspectos relacionados con el modo multijugador.

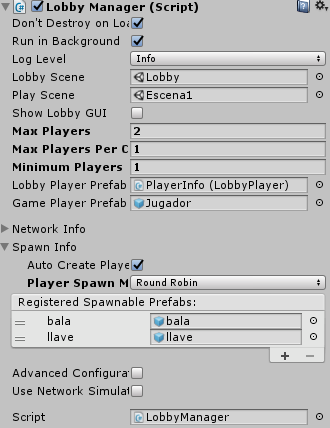


Figura 10. Componente Manager UNET

Este componente se añade a un GameObject del menú. En él se pueden configurar la escena “Lobby” y en la opción *Play Scene* es donde se añade la escena en la cual se van a conectar los jugadores.

Como se ve en la Figura 10, se pueden configurar el máximo de jugadores (en este caso 2), así como el mínimo de jugadores que será 1.

Otro dato importante que tenemos que configurar en esta parte es añadir el *prefab* del jugador en la pestaña *Game Player Prefab*, para que cada jugador controle a su propio personaje dentro de la escena.

En la opción de “*Spawn info*” se añaden los *prefabs* que se instancian en ejecución dentro de la escena para que solo haya una instancia del objecto tanto en el cliente como en el servidor.

Todos los GameObjects que esten en línea deben de llevar el componente:



Figura 11. Componente Network Identity

Este componente sirve para diferenciar a los objetos dentro de la escena multijugador.

Otra información importante que hay que enviar al servidor para el correcto funcionamiento de la opción *Multiplayer* son la posición del objecto en la escena y las animaciones que está realizando en ese momento. Estos componentes son *Network Transform* (para enviar la información del movimiento del objecto) y *Network* *Animator* (para la información de las animaciones).

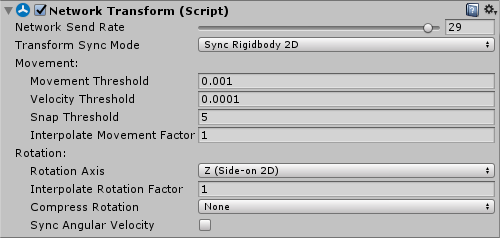


Figura 12. Network Transform

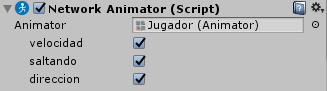


Figura 13. Network Animator

En el componente *Network Animator*, activando las opciones de los parámetros de las animaciones que aparecen se envía la información al servidor.

### EXPLICACIÓN DE UNITY

A continuación, se procede a describir el funcionamiento del motor gráfico Unity, así como de resaltar algunas de sus características de las aplicaciones multiplataforma que es posible llevar a cabo mediante su editor y su scripting.

El aprendizaje del motor gráfico comenzó en la asignatura optativa de “videojuegos” del segundo semestre del 4ºcurso. Durante este tiempo se aprendieron la parte más básica, con la cual comenzar a crear pequeños proyectos y sirvió para familiarizarse con el entorno de trabajo. Esto proporcionó los conocimientos para que, mediante la documentación, foros y tutoriales, de forma autodidacta profundizar mucho más en el funcionamiento del motor y adquirir los conocimientos que a día de hoy tengo.

Lo primero que aparece una vez que se arranca el programa es la interfaz principal. Esta está dividida en varias partes, las cuales se pueden distribuir de varias formas propuestas por el software o crear el propio usuario una con la que esté cómodo. Busca ser intuitiva, separando las escenas de juego y edición, y dando la posibilidad de arrastrar los objetos entre los diferentes componentes dando una sensación de unidad.

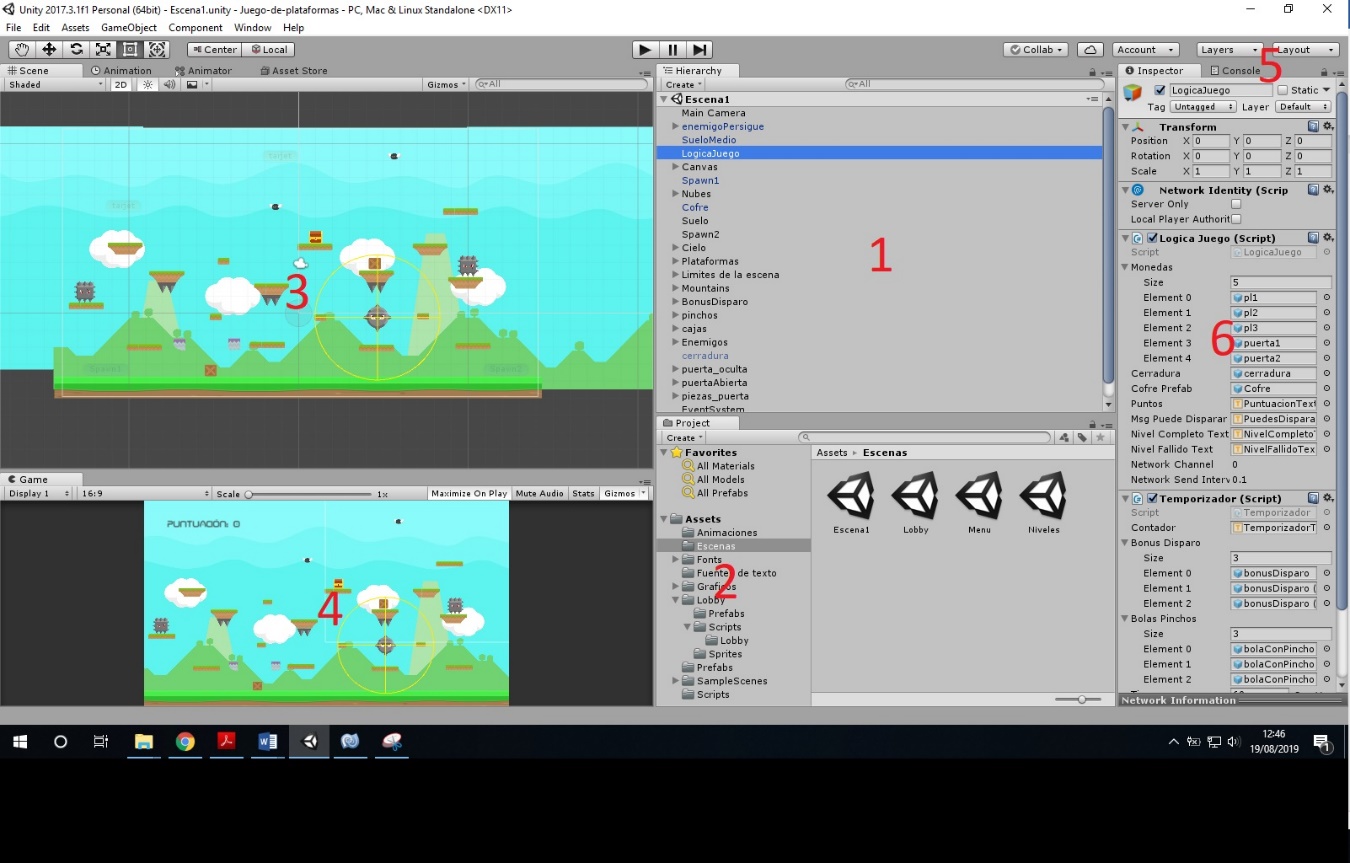


Figura 14. Distribución layout Unity

* **1- Hierarchy:** Este componente es una representación de texto jerárquico de cada objeto en la escena. La jerarquía revela la estructura de cómo los objetos están agrupados y las herencias que hay entre ellos.
* **2- Project Window:** En esta ventana aparecen los diferentes assets del proyecto, por convenio se dividen en diferentes carpetas (Sprites, scripts, materiales, etc.).
* **3-** **Scene View:** Esta vista está pensada para editar el proyecto, permite la navegación visual. Se puede configurar para trabajar tanto en 2 como 3 dimensiones.
* **4- Game View:** En esta vista se renderizan los objetos creados en la vista Scene View mediante los enfoques de los objetos tipo cámara. Muestra el resultado que aparecerá en la aplicación final.
* **5- Console Window:** Este componente muestra los mensajes mandados desde el scripting, errores y avisos. Es utilizada por los desarrolladores.
* **6- Inspector:** En este componente se visualizan los diferentes componentes, características y propiedades del objeto seleccionado. Desde aquí se podrán modificar los valores y editar los componentes asociados a cada uno de los objetos.

Antes de seguir hay que dejar claros algunos conceptos dentro de este motor:

* GameObjects: Son los elementos que interactúan entre ellos en el juego. Es cualquier objeto existente dentro de la jerarquía de Unity. Es la parte principal de Unity ya que sin estos no existen los videojuegos.
* Scripst: Proporcionan funcionalidades a los GameObjects mediante diferentes lenguajes de programación. El más extendido para este motor es C#. Estos archivos de texto se encargan de las funcionalidades, características o comportamientos de los GameObjects.



Figura 15. Estructura del funcionamiento de Unity

## C#

Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows Microsoft, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux [7].

## GITHUB

GitHub es un repositorio donde he podido ir almacenando los cambios que se han ido haciendo según se avanzaba con el desarrollo del proyecto. Cuando se hacían cambios importantes en el proyecto realizaba un commit con los cambios.

A modo de copia de seguridad y control de versiones es muy útil para este tipo de proyectos.

## FREE PLATAFORM GAME ASSETS

Para los *sprites* del proyecto he utilizado un *asset* que proporciona Unity de forma gratuita, donde se encuentran diversos gráficos de enemigos, plataformas, personajes, objetos, etc…[ASSET].

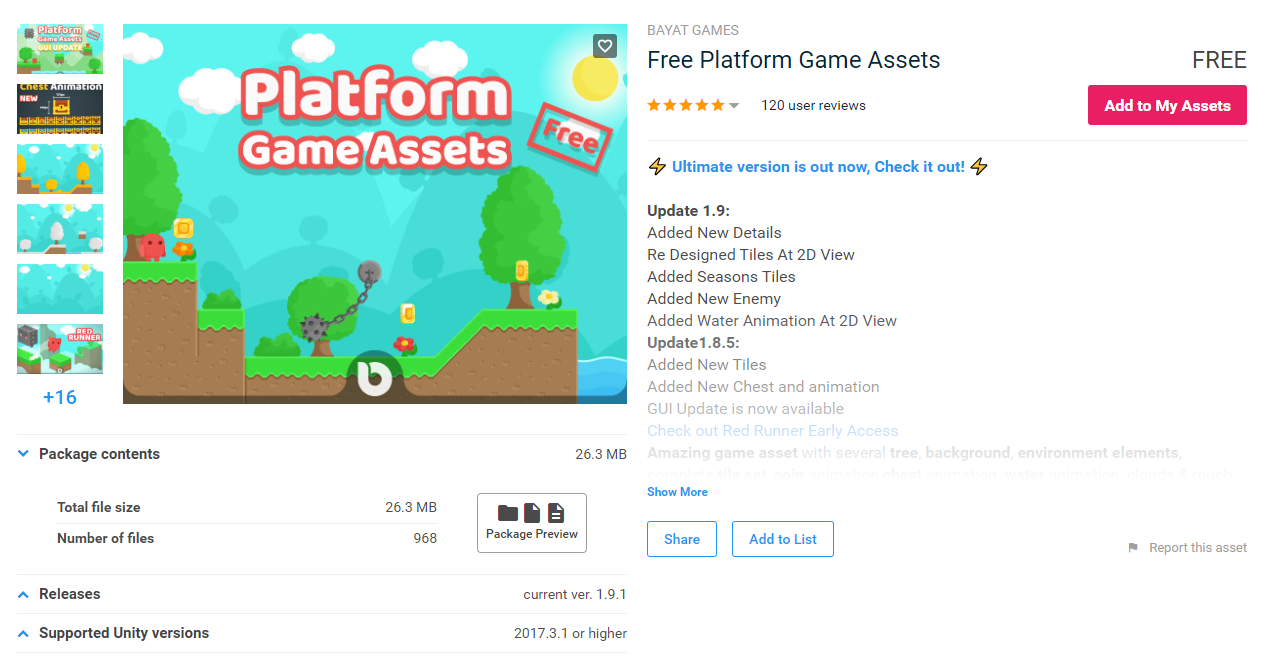


Figura 16. Free plataform game assets

# ASPECTOS RELEVANTES DEL DESARROLLO

En este punto se van a destacar los aspectos más importantes para el desarrollo del proyecto desde la perspectiva del diseño e implementación.

El tema del proyecto fue propuesto al tutor para realizar el Trabajo Fin de Grado. Después de comentar la propuesta del trabajo, se dio el visto bueno para realizar el proyecto.

En febrero de 2019, se presenta la propuesta de Trabajo de Fin de Grado en la secretaría de la Escuela Politécnica Superior de Zamora (EPSZ).

En marzo de 2019, el tema fue adjudicado y aprobado por la Universidad de Salamanca.

## PLANTEAMIENTO DEL PROYECTO

Desde un principio supe que quería realizar como trabajo de fin de grado un videojuego ya que me gustó bastante la asignatura de “Videojuegos” del grado y me pareció muy interesante. Se valoraron diferentes opciones de que tipo de juego realizar: juego por turnos, *RPG*, etc. Pero finalmente la idea que me convenció fue un juego de plataformas que para añadir complejidad a la idea si impuso el tema de la cooperación entre 2 jugadores para lograr el objetivo del juego.

Una vez escogida la idea que quería desarrollar se lo comenté a mi tutor Juan Carlos Matos Franco, le pregunté su opinión sobre la viabilidad y complejidad del proyecto que me dio el visto bueno y accedió a tutorizar mi proyecto.

## METODOLOGÍA DEL CICLO DE VIDA DEL SOFTWARE

Se puede definir ciclo de vida del software como “Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando la vida del sistema desde la definición de requisitos hasta la finalización de su uso” [ISO 12207] (2017). Mediante este ciclo de vida se pretenden representar las actividades y productos intermedios que son necesarios para el desarrollo de una aplicación.

Existe una gran variedad de procesos de software a seguir para las diferentes actividades que tienen lugar durante un proceso. Pero independientemente del proceso escogido se caracterizan por una serie de fases generales:

* **Análisis:** Fase inicial del proyecto donde se decide qué hacer, cómo hacerlo, las herramientas a emplear y los objetivos del proyecto.
* **Diseño:** Fase en la que se lleva a cabo la definición de las interfaces, navegación de la aplicación, así como de la arquitectura y otras características de un sistema o componente que nacen de este proceso.
* **Implementación:** Fase del proceso en la cual se desarrolla el código necesario para la creación de la aplicación, se implementa la funcionalidad a la aplicación.
* **Pruebas:** Fase en la cual se comprueban las funcionalidades implementadas en la fase anterior.

**Mantenimiento:** Fase la cual comprende todas las actualizaciones secundarias de software (pertenecientes al proceso de mantenimiento continuo), así como todos los procedimientos correctivos del software (pertenecientes al proceso de mantenimiento continuo).



Figura 17. Ciclo de vida del software

### MODELO ITERATIVO INCREMENTAL

Atendiendo al alcance del ciclo de vida del software, y las características de las fases que dividen el ciclo, se decide emplear **el Modelo iterativo e incremental**, ya que se permite crear versiones completas de software en cada iteración. Se selecciona este modelo ya que los requisitos no están todos establecidos inicialmente, y se pueden ir cambiando y añadiendo funcionalidades a la vez que se dispone de una versión funcional mejorable del producto.

Los pasos clave en este proceso son iniciar con una implementación simple de los requerimientos iniciales del sistema, y a medida que seguimos en iteraciones posteriores, mejorar la secuencia evolutiva de versiones hasta que el sistema queda completamente implementado (en cada iteración se realizan cambios en el diseño y se agregan funcionalidades nuevas y capacidades al sistema).

De esta forma, el desarrollador del videojuego no sabe inicialmente qué es lo que se necesita para satisfacer las necesidades de una manera clara, y en el desarrollo debido a que surgen problemas y cambios los procesos tienden a cambiar.

Tabla 1. Ventajas e inconvenientes del modelo iterativo e incremental

|  |  |
| --- | --- |
| VENTAJAS | INCONVENIENTES |
| Se dividen en subsistemas que están de acuerdo con la funcionalidad. | Al entregar de forma temprana los proyectos simples, es un poco monótono el trabajar con pequeños cambios. |
| Las versiones quedan definidas comenzando con un subsistema inicial pequeño, al que se irá agregando funcionalidad con cada nueva versión. | Puede suponer una excusa o una forma de saltarte el tener que planificar y gestionar un proyecto desde el primer momento. |
| Los productos desarrollados así, tienen menor probabilidad de fallar, y favorece el añadir nuevas necesidades. | Cada secuencia lineal supone un incremento, y el ciclo de vida iterativo puede engendrar problemas debido a esto. |
| Aumento del aprendizaje en cada iteración que aumenta la reutilización para posteriores proyectos. | Puede suponer una falta de robustez en el sistema las múltiples entregas de un programa parcial pero funcional. |



Figura 18. Funcionamiento gráfico del Modelo Iterativo e Incremental

## PLANIFICACIÓN DEL PROYECTO

Una vez que se obtuvo la adjudicación del proyecto definitivo (22 febrero de 2018) se inicia la planificación del proyecto, aunque con anterioridad ya se habían empezado a tratar algunas ideas con el tutor ya que este esperaba que el tema fuese aceptado.

Como se vio en el anterior apartado, se elige el proceso iterativo e incremental como modelo de ciclo de vida del software.

Siguiendo la metodología de trabajo escogida se irá desarrollando la aplicación a la vez que se van documentado los posibles cambios que se produce en cada iteración, con el finde de que el crecimiento del proyecto final y de la memoria sean al unísono y de una manera uniforme. Facilitando el proceso al final, tan solo juntando las partes.

Aparte de utilizar *GitHub* como control de versiones también he utilizado la herramienta *Google Drive* a modo de “*Backup*” en caso de pérdida o borrado del proyecto tener un respaldo.

## FASES ITERATIVAS

El proyecto finalmente se concretó en tres iteraciones, en cada una de las cuales se implementaron nuevas funcionalidades.

* **Iteración** **1**:
  + **Análisis**: Se definió el estilo visual que se desarrollará en el videojuego, la estética y el diseño de nivel genérico. Se definen las primeras funcionalidades del personaje.
  + **Diseño**: Familiarización con el entorno de trabajo del motor gráfico diseño del primer nivel y primeros enemigos.
  + **Implementación**: Creación de la estructura del primer nivel y movimiento del jugador y algunos enemigos y su implementación.
  + **Pruebas**: Las pruebas serán de los apartados concernientes a esta primera iteración.
  + **Documentación**: Correspondiente a la primera iteración.
* **Iteración** **2**:
  + **Análisis**: Definición de las opciones que tendrá disponible el usuario. *Puntuación simbólica* del jugador.
  + **Diseño**: Creación de nuevos enemigos. Diseño de una simple inteligencia artificial a uno de ellos.
  + **Implementación**: Se implementan los disparos del jugador, así como la salud del jugador y otros aspectos relevantes de los enemigos.
  + **Pruebas**: Prueba de jugabilidad y comprobación de la adaptación de las nuevas funcionalidades con la versión anterior.
  + **Documentación**: Correspondiente a la segunda iteración.
* **Iteración** **3**:
  + **Análisis**: Análisis de los componentes de la interfaz de Usuario del juego y análisis del balance del juego.
  + **Diseño**: Creación componentes para la interfaz de usuario y diseño de nuevos niveles.
  + **Implementación**: Implementación de la interfaz final, creación de menús y nuevos niveles.
  + **Pruebas**: Prueba de jugabilidad y comprobación de la adaptación de las nuevas funcionalidades con la versión anterior, y balance de juego.
  + **Documentación**: Correspondiente a la tercera iteración.

## IMPLEMENTACIÓN

En este punto se explicarán los algoritmos utilizados, detallando las estructuras de datos utilizadas. Como ya se comentó anteriormente el lenguaje de programación utilizado para desarrollar la aplicación es C#.

### DISTRIBUCIÓN DE LOS ASSETS

Una vez completado el proyecto la distribución final de los *assets* fue la siguiente:



Figura 19. Distribución de los Assets en Unity

* ***Juego cooperativo de plataformas en 2D****:* Proyecto en Unity
  + **Assets:** Carpeta raíz que contiene todos los *assets* que podrán ser utilizados en el proyecto de Unity
    - **Animaciones:** Esta carpeta contiene todas las animaciones y los esquemas de animación del proyecto.
    - **Scenes:** En este directorio se encuentran las diferentes escenas que se irán cargando paulatinamente durante el desarrollo del juego.
    - **Fuentes de texto:** Es esta carpeta se encuentran las diferentes fuentes caligráficas utilizadas en el proyecto.
    - **Gráficos:** Contiene las imágenes que se han utilizado en el juego, por ejemplo, *sprites* de los personajes, enemigos, plataformas...
    - **Importaciones:** En este directorio se encuentran los recursos importados desde terceros, como pueden ser materiales o modelos.
    - **Prefabs:** En esta carpeta podemos encontrar los diferentes *prefab* utilizados en el proyecto.
    - **Scripts:** Esta carpeta contiene los scripts que dan las funcionalidades a la aplicación.

### 5.1.2. SCRIPTS

El proyecto se ha estructurado de tal forma donde podemos identificar 6 scripts como los más importantes: ‘LogicaJuego’, ‘MenuControl’, ‘menu’, ‘movimiento’, ‘temporizador’ y ‘persigueJugador’ (en el cual se implementa una simple inteligencia artificial). Éstos se explicarán a continuación de forma más detallada.

Por otro lado, tenemos el resto, los cuales contienen las funcionalidades propias de cada objeto las cuales se harán referencia a través de llamadas o instancias desde otros scripts.

El árbol de scripts es el siguiente:

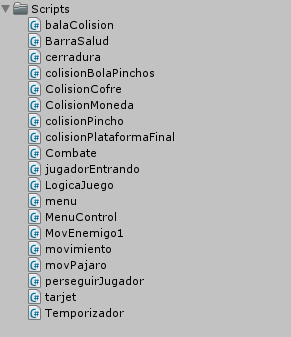


Figura 20. Árbol de scripts del proyecto.

Ahora se hará un desarrollo más completo de los scripts más relevantes:

* *LogicaJuego:* Script encargado de controlar todas las lógicas del juego.



Figura 21. Método Update de la LogicaJuego

En este método analizamos el funcionamiento del juego:

* **La puntuación simbólica**: puntuación que ganan los jugadores a lo largo del juego, se dice simbólica ya que al ser un juego cooperativo dicha puntuación no sirve para ganar al otro jugador sino a modo de información, ya que el objetivo será otro.
* **El control de disparo del jugador**: El jugador solo podrá disparar en ciertos momentos de la partida, cuando alcance una serie de objetos.
* **La aparición de objetos**: Parte muy importante del juego, según se van eliminando objetos clave para el juego, van apareciendo otros en la escena.
* **Comprobación del nivel completado**: En este método se comprueba que los jugadores hayan superado el nivel.

Este script tiene una serie de propiedades estáticas a las cuales se puede acceder desde otros scripts para controlar todo tipo de lógicas del juego:

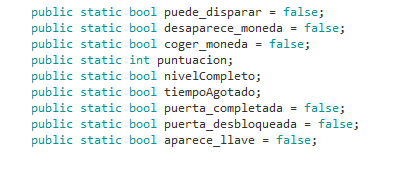


Figura 22. Propiedades estáticas de LogicaJuego (Script)

* *Menu*: Script donde se encuentran los métodos para cambiar entre escenas.



Figura 23. Gestor de escenas (Script)

* *Temporizador:* Script muy importante en el desarrollo del juego ya que en el se controla el tiempo de juego en la escena. Es decir, el tiempo que tiene el jugador o jugadores en caso de multijugador en lograr el objetivo. En caso de que sea multijugador ese tiempo tiene que estar sincronizado para ambos jugadores, eso se hace mediante este script:

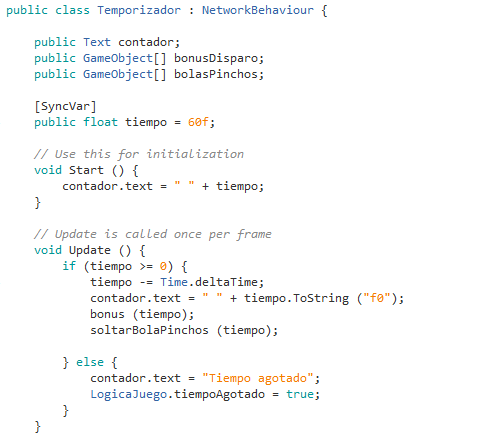


Figura 24. Fragmento Temporizador (Script)

Mediante la terminología [*SyncVar*] (proporcionada por la herramienta *UNET*) se hace que esa variable se sincronice con el servidor, de esta manera el contador siempre tendrá el mismo valor tanto en el cliente como en el servidor.

* *Movimiento*: Script que controla el movimiento del personaje, la opción de disparo del jugador, entre otras.



Figura 25. Movimiento del jugador (Script)

Como se puede ver en la Figura 25, el movimiento del jugador se basa en lo siguiente: cuando se pulsan las teclas de dirección se le aplica una fuerza al objeto para activar su movimiento. En el caso de pulsar la barra espaciadora, la fuerza hace que el personaje salte.

Al principio de la función aparece el siguiente fragmento de código:



Figura 26. Fragmento de código de movimiento (Script)

Esta opción proporcionada por la herramienta *UNET* hace que el movimiento solo se realice en un jugador, es decir, si se conectan 2 jugadores a la partida que cada jugador controle a su propio personaje.

* *MenuControl*: Script donde se controla el inicio del juego multijugador. En él se encuentran métodos para la conexión de la partida.



Figura 27. MenuControl (Script)

* *persigueJugador*: Script donde se implementa una simple inteligencia artificial aplicada a uno de los enemigos. Esta inteligencia artificial consiste en que cuando el jugador se acerca a una distancia del enemigo, éste persigue al jugador con la intención de quitarle salud hasta que el jugador sale de su rango de acción donde el enemigo regresa a su posición de origen.

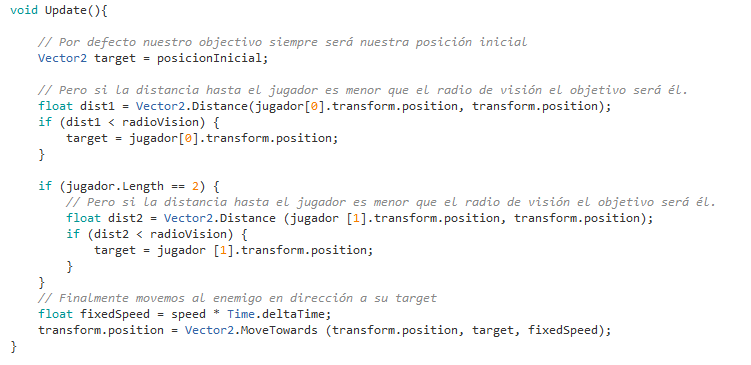


Figura 28. Inteligencia artificial del enemigo (Script)

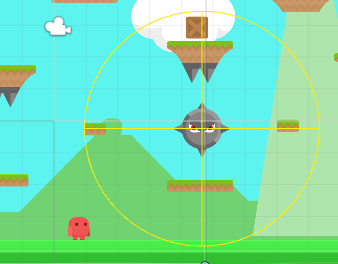


Figura 29. Rango de acción del enemigo (Escena)

# PROBLEMAS ENCONTRADOS

Durante el desarrollo del proyecto han surgido una serie de problemas los cuales algunos se han ido solucionando y otros perduran.

A continuación, se exponen algunos de estos problemas, así como las soluciones dadas para poder continuar con el correcto desarrollo del proyecto.

## DESCONOCIMIENTO DEL MOTOR DE JUEGO Y EL LENGUAJE DE PROGRAMACIÓN

Para poder hacer frente a este proyecto y el volumen de trabajo que conlleva ha sido necesario familiarizarse con el entorno de desarrollo de Unity3D, así como con el lenguaje de programación C#. Para estos objetivos se han realizado múltiples búsquedas en foros páginas web.

Uno de los puntos más favorables es la gran comunidad que está detrás de este motor gráfico, donde la gente está muy dispuesta a ayuda, además al llevar unos cuantos años funcionando muchas de las dudas que pueden surgir ya están resueltas en foros.

Por último, el lenguaje C# no es de una sintaxis muy compleja. Al ser orientado a objetos la forma de trabajo es parecida a Java, del cual se tenían algunos conocimientos previos, lo cual ha ayudado en gran medida a la hora de entender algunas de las funciones.