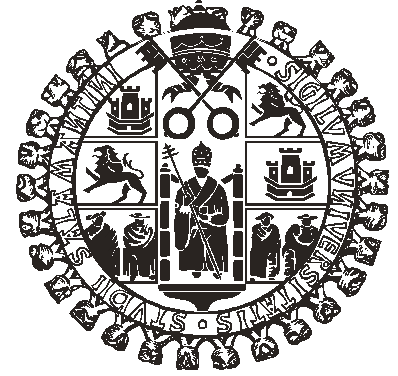
UNIVERSIDAD DE SALAMANCA

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 DE ILUSTRACIONES

**No se encontraron entradas de tabla de contenido.**

# 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.

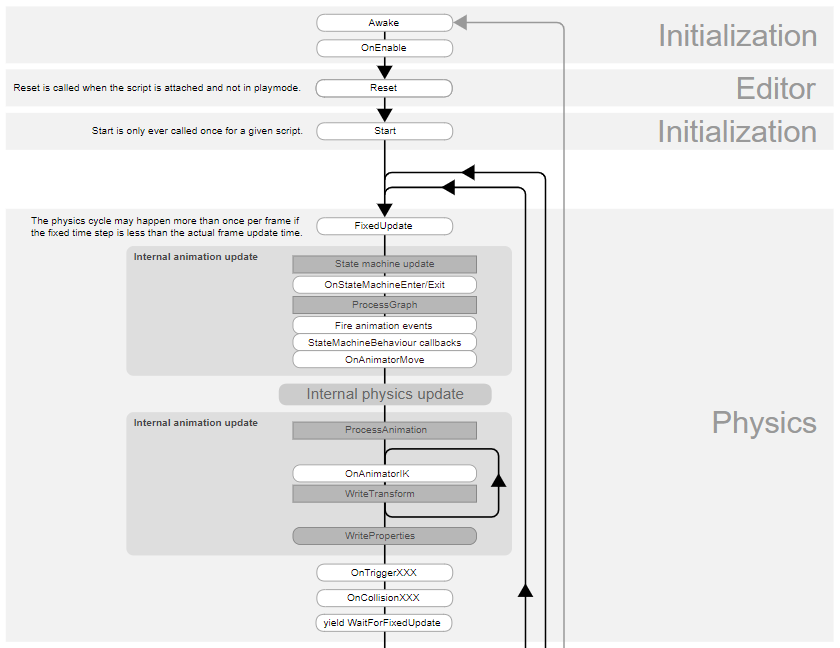


Figura 1. Primer tramo del ciclo de vida

### 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.

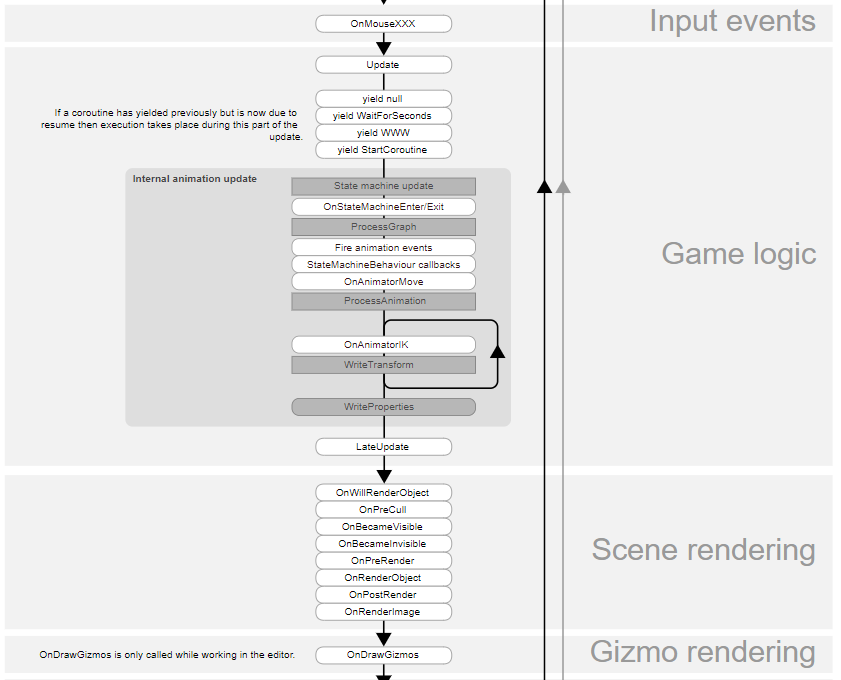


Figura 2. Segundo tramo de ciclo de vida

### 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.

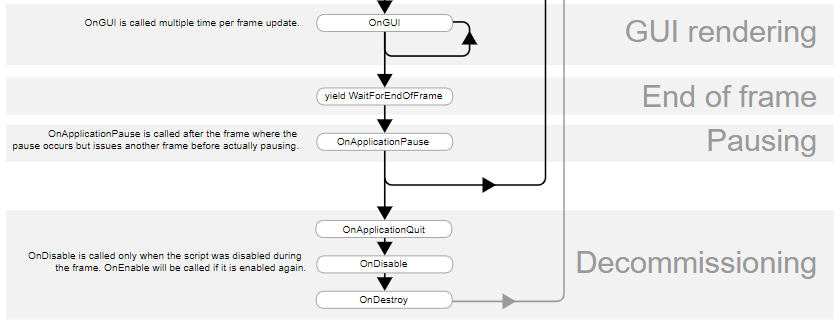


Figura 3. Tercer tramo de ciclo de vida

## UML Y LOS DIAGRAMAS DE ESTADOS