The Legend of the Arrow

Sergio Pérez Cuevas

Resumen—The Legend of the Arrow es un proyecto que engloba todas las fases de la creación de un videojuego estilo plataformas en 2 dimensiones. Para la realización del proyecto se ha seguido la metodología SUM, que es la más comúnmente empleada para desarrollos de este tipo. El objetivo principal de este proyecto es familiarizarse con el entorno de desarrollo de videojuegos e incluir técnicas de inteligencia artificial para el comportamiento de los personajes no controlados por el jugador. Este videojuego ha sido desarrollado dentro del entorno de programación UNITY usando el lenguaje C# para programar los scripts necesarios para el funcionamiento del mismo. Además de las tareas de programación de las diferentes entidades que aparecen en el juego, también se han llevado a cabo tareas como el diseño de interfaz y apariencia de los elementos del juego. El resultado final de este proyecto es una versión completamente funcional del juego que cumple con todos los requisitos especificados al comienzo de la planificación del proyecto.

Palabras clave— Videojuegos, Plataformas 2D, proyectil, IA, UNITY, script, assets, gameObject.

Abstract— The Legend of the Arrow is a project that encompasses all phases in the creation of a 2D platform videogame. SUM methodology has been selected to guide the development of this project, as it is commonly used for developments of this kind. The main objective of this project is to familiarize with game development environment and to include artificial intelligence techniques for the behavior of non-player controlled characters. This game has been developed within the context of UNITY programming using the C # language to program the scripts required for its correct functioning. Besides programming tasks of the various entities that appear in the game, tasks such as interface design and appearance of the elements of the game have also been carried out. The end result of this project is a fully functional version of the game that meets all the requirements specified at the beginning of the project planning.

Keywords— Videogames, Platforms 2D, projectile, IA, UNITY, script, assets, GameObject.

1 Introducción

En esta primera sección se expondrán las motivaciones principales que han llevado a la realización del proyecto y se detallarán tanto los objetivos a alcanzar como la planificación establecida para poder llevarlos a cabo.

1.1 Motivación

Hoy en día la informática está presente en muchísimos aspectos y ámbitos en la vida de cualquier persona, desde smarthphones, el ordenador personal de casa o un panel en un sistema domótico. Y dentro de este campo, los videojuegos son considerados un entorno de negocio importante a nivel creativo como a nivel económico.

Como muestra de su creciente auge, más de un 25% de los habitantes de Europa (unos 95 millones aproximadamente) son usuarios de videojuegos. Esto supone un volumen de negocio de 18.500 millones de euros (31,3% del mercado mundial [1].

Considerando esta coyuntura, la motivación principal para la realización del proyecto es poder familiarizarse con todas las tareas relacionadas con la programación de un videojuego, incluyendo aspectos tanto de programación como de diseño.

Con el fin de poder alcanzar resultados dentro del tiempo disponible se ha seleccionado realizar un videojuego del tipo plataformas 2D. La selección de este género viene dada porque nos permite desarrollar todas las fases del videojuego a un nivel de complejidad asumible. Este género consiste en desplazar un personaje por un terreno. El personaje interactúa tanto con el terreno como con otros elementos del mismo, como pueden ser otros personajes dotados de diversos niveles de inteligencia artificial (IA), desde IA muy simple a IA más compleja.

Por tanto, las tareas principales que habrá que completar serán: diseño del terreno y de los personajes, programación de las interacciones entre el personaje principal y otros elementos y desarrollo de la interfaz de usuario.

1.2 Objetivos a alcanzar

Los objetivos principales que se proponen para la finalización del proyecto son

- Diseño e implementación de la interfaz de usuario.
- Desarrollo del terreno.
- Introducción de personaje jugable y personajes dotados de inteligencia artificial controlados por el computador.
- Interacción de los personajes con el terreno.
- Diseño de todos los personajes.

Además de estos objetivos, se añaden los siguientes:

- Introducción de música y sonidos.
- Diseño de proyectiles.
- Creación de checkpoints.

• Definición de sistema de puntos.
• Añadir más escenas y niveles de juego.
• Desarrollar niveles adicionales de inteligencia artificial.

1.3 METODOLOGÍA A SEGUIR

La metodología seleccionada para el desarrollo del proyecto ha sido SUM [2], la cual se explicará en detalle en la sección 2.4.

1.4 REQUISITOS DEL PROYECTO

Los requisitos del proyecto son:

1.4.1 Interfaces externas

La pantalla inicial contendrá una ventana con el título del juego, un fondo ambientado en el videojuego y un botón para iniciar el juego. Al pasar por encima del botón de inicio, éste cambiará de color para que el usuario sepa que el mouse está en el sitio correcto.

Los objetivos principales planteados para el desarrollo del software deben poder ser alcanzable y la interfaz de usuario debe de ser lo más simple posible evitando funcionalidades no necesarias.

1.4.2 Requisitos funcionales

El juego ofrecerá al usuario un conjunto de pantallas que deberán ser completadas sorteando una serie de obstáculos y luchando contra enemigos controlados mediante inteligencia artificial. Una vez finalizada una pantalla, el sistema pasará automáticamente al siguiente nivel.

El juego debe permitir al usuario mover al personaje principal e interactuar con el menú principal, donde podrá empezar juego nuevo.

1.4.3 Requisitos de rendimiento

Dado que el proyecto consiste en un videojuego 2D y se pretende que sea exportable a diversas plataformas además de PC se diseñará con el objetivo de consumir pocos recursos.

1.4.4 Escalabilidad

El juego ha de ser directamente escalable lo cual es asequible de conseguir ya que siendo un juego de plataformas, es posible ampliar la historia, añadir nuevas pantallas o enemigos sin tener que añadir nuevas capas de desarrollo.

1.5 PLANIFICACIÓN INICIAL DEL PROYECTO

En esta sección se detallarán las tareas que se han seguido para realizar el proyecto, también los recursos utilizados y los posibles riesgos.

1.5.1 Lista de tareas

Las tareas que definieron al inicio del proyecto son las siguientes:

- Captura de requisitos.
- Toma de contacto con Unity.
- Diseño Interfaz Usuario (UI).
- Implementación inicial UI.
- Desarrollo del terreno Unity.
- Diseño personaje principal.
- Introducción del personaje principal.
- Interacción del personaje principal con el entorno.
- Funcionalidad Menú.
- Diseño de la Inteligencia Artificial.
- Diseño NPC's.
- Introducción Personajes no Controlados por el Jugador (NPC's).
- Introducción de música y sonidos.

Durante el desarrollo del proyecto se ha visto necesario incluir otras tareas adicionales:

- Seguimiento de cámara.
- Introducción de checkpoints.
- Definición de un sistema de puntuación, control de la vida y daño del personaje.
- Mejoras en interfaz: introducción de texto emergente.
- Introducción de proyectiles y diseño de su interacción con el personaje.
- Definición de límites de la pantalla.

1.5.2 Recursos disponibles

Los recursos utilizados para la realización del proyecto han sido en su mayor proporción los relacionados con la herramienta Unity [3]. Mediante Unity se ha podido generar la pantalla, añadir componentes, generar scripts [4] a partir de MonoDevelop incluido en Unity y generar las escenas necesarias para ejecutar el videojuego.

También ha sido necesario el uso de GIMP [5], un editor de imágenes, para generar algún asset o diseño de algún elemento del videojuego.

1.5.3. Riesgos del proyecto

Al comenzar a desarrollar dentro de un entorno nuevo y un lenguaje de programación que no había sido utilizado previamente, los riesgos iniciales de desarrollo eran más numerosos. El punto clave del proyecto consiste en el desarrollo de la IA, la cual se dividió en tres niveles en base a su dificultad. Durante la realización del proyecto se añadieron otros objetivos para mejorar la calidad del videojuego como la música o los proyectiles.

1.6 ESTRUCTURA DEL DOCUMENTO

La estructura del artículo es la siguiente: en el apartado de introducción se definen objetivos del proyecto y requisitos recogidos para el desarrollo del mismo. La siguiente sección es el estado del arte donde se hace una breve introducción histórica a los videojuegos y en especial al género plataformas, también se explica de forma

concisa un motor de videojuego y la metodología SUM. La sección diseño del proyecto explica como se ha hecho, incluyendo ilustraciones explicativas. El documento finaliza detallando el estado actual del juego y con una sección de conclusiones donde se reflexiona sobre los conocimientos adquirido.

2 ESTADO DEL ARTE

En esta sección se explicará la historia de los videojuegos y en especial los videojuegos del género plataformas, también se detallarán las herramientas de desarrollo más extendidas actualmente y con especial énfasis en Unity, que ha sido la herramienta utilizada para llevar a cabo este proyecto. Por último se hará una breve explicación de la metodología de desarrollo escogida para realizar el videojuego.

2.1 BREVE HISTORIA DE LOS VIDEOJUEGOS

En cuanto empezaron a crearse las primeras supercomputadoras en la década de los 40, enseguida se vio la oportunidad de implementar juegos como el ajedrez para poder ser jugados en dichas computadoras. No fue hasta 1960 cuando se dio un avance significativo y durante esta década, grupos de estudiantes o apasionados de la computación desarrollaron juegos como spacewar o la brownbox (la cual incorporaba varios juegos) [6].

Durante la década de los 70, se crearon las primeras máquinas recreativas, como Galaxy Game -la primera máquina arcade de la historia [6]- o Magnavox Odissey [6], considerada como la primera videoconsola de la historia.

La época dorada de los videojuegos llegó la década de los 80, tras décadas de desarrollo a nivel computacional de juegos que utilizaban algoritmos o juegos adaptados a la tecnología del momento. En esta década empezaron a forjarse grandes empresas que actualmente monopolizan el mundo de los videojuegos como Nintendo o Sega. En este sentido Japón es el país considerado como la cuna del videojuego actual y representa unos de los mercados más poderosos que hay en el mundo.

Desde 1990 aproximadamente las mejoras tanto en nivel de programación como diseño son cada vez más frecuentes y el mundo de los videojuegos está en continuo desarrollo, surgiendo nuevos géneros como los juegos realistas en tres dimensiones o incluso aquellos basados en realidad virtual.

2.2 BREVE CATEGORIZACIÓN/HISTORIA SOBRE LOS PLATAFORMAS 2D

El género de videojuegos el cual se ha empleado en este proyecto (plataformas) empezó a popularizarse en los años 80. Dicho género se caracteriza por tener a un personaje el cuál tendrá que caminar, correr o saltar sobre una serie de plataformas con enemigos, mientras se recogen objetos para completar el juego.

Los videojuegos plataformas nacen como tal sobre 1980, a partir de las máquinas arcade con videojuegos como Space Panic o Donkey Kong. Una de las características más importantes que hizo diferenciarse del resto de videojuegos fue la gravedad que siente el objeto que controlamos al saltar o desplazarse por la pantalla.

En 1985 apareció el juego más determinante del género, Super mario bros de la compañía nipona Nintendo. En 1990, con la llegada de videoconsolas portátiles como Game Boy, empezaron a hacerse muy populares los videojuegos de plataformas introduciendo nuevos sistemas de inteligencia artificial para hacerlos más entretenidos y retantes para el jugador.

A día de hoy, ha habido una evolución en la que los videojuegos de plataformas, que han pasado a desarrollarse en tres dimensiones. La complejidad creciente de estos juegos ha supuesto un coste de desarrollo mayor, lo cual, considerando una pérdida en popularidad del género de plataformas 3D, hizo que este género fuera minoritario durante la década de los 2000. El auge de las herramientas de desarrollo de videojuegos a bajo coste y complejidad y la financiación de videojuegos por internet (crowfunding) ha hecho que muchos desarrolladores noveles hayan visto la oportunidad de desarrollar videojuegos simples de una manera más rápida, lo cual ha conllevado la recuperación de géneros antiguos como los plataformas 2D.

Dentro del campo de plataformas 2D, los videojuegos más importantes del género en los últimos 30 años han sido: la saga de Super Mario, Rayman, Castlevania, Sonic, Donkey Kong, o Metroid, la mayoría usan scroll horizontal/lateral.

2.3 MOTOR DE VIDEOJUEGO

Hoy día existen infinidad de herramientas de desarrollo de videojuegos tales como Shiva3D [7], GameMaker Studio [8], Torque2D [9] o Unity [3] siendo estos los más utilizados y extendidos.

A continuación se detallará el concepto un motor de videojuego. Un motor de videojuego es un entorno de programación que permite el diseño, la creación y la representación de un videojuego.

A continuación se enumeran los elementos más importantes de un motor de videojuego:

- Assets: Son elementos que se introducen al videojuego, desde modelos del terreno o personaje hasta scripts o sonidos.
- Render: Es el proceso de la computadora en mostrar en pantalla el aspecto visual del videojuego.
- Inteligencia Artificial: Para todo videojuego es importante la inteligencia artificial de la cual será dotado, desde IA muy simple hasta IA sumamente compleja.

Para este proyecto se ha utilizado el motor de videojuego Unity 4.5.

Unity es un entorno multiplataforma y se pueden desarrollar videojuegos en los sistemas operativos Windows y OS X en los cuales se permite crear juegos para Windows, OS X, Linux, Xbox 360, Playstation 3, Playstation Vita, Wii, Wii U, iPad, iPhone, Android y Windows Phone. También es el motor de videojuego con más documentación y soporte disponible de los mentados anteriormente haciendo que esto sea un punto determinante para su

elección final.

Ha sido de gran ayuda para este proyecto el uso de la store de Unity [9], el cual es un recurso disponible en el editor de Unity que incluye modelos 3D, texturas y materiales, música, sistemas de partículas etc. a disposición del usuario, aunque la gran mayoría de pago.

Unity lleva integrado un entorno de desarrollo, MonoDevelop, con el cual se ha podido desarrollar todo el scripting del videojuego.

2.4 METODOLOGÍA SELECCIONADA

La metodología SUM [2] para videojuegos tiene como objetivo desarrollar videojuegos de calidad en tiempo y costo, así como la mejora continua del proceso para incrementar su eficacia y eficiencia. Dicha metodología posibilita obtener resultados predecibles, administrar eficientemente los recursos y riesgos del proyecto, y lograr una alta productividad del equipo de desarrollo, aunque para este proyecto no ha habido equipo como tal, ya que ha sido una persona el encargado del proyecto en su totalidad.

SUM fue concebida para adaptarse a equipos multidisciplinarios pequeños y para proyectos cortos con alto grado de participación del cliente, en este caso el tutor del proyecto.

Se han seguido cinco fases para realizar el proyecto: concepto, planificación, elaboración, beta y cierre, tal y como podemos ver a continuación en la figura 1:

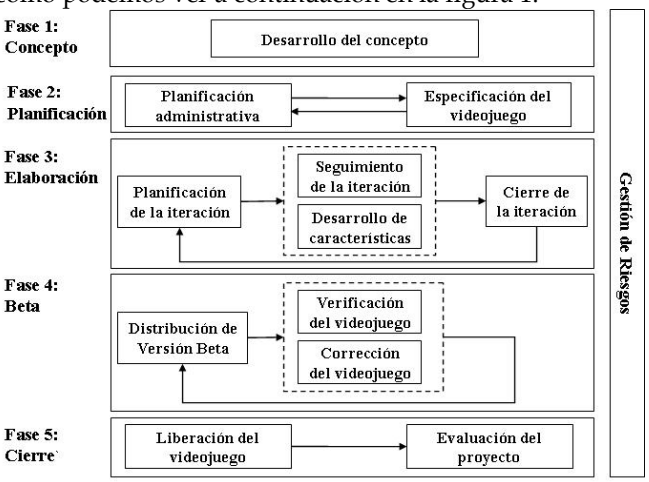


Figura 1: Fases del proceso

A continuación se describen las 5 fases del proyecto:

- En la fase 1, concepto, se ha definido el concepto de videojuego, la temática del mismo, herramientas a utilizar y principales características que contendrá el proyecto.
- En la fase 2, planificación, se han creado una serie de tareas mediante un diagrama de Gantt cada una de ellas con una duración temporal.
- En la fase 3, elaboración, se ha implementado el videojuego con una metodología basada en tres pasos. El primer paso ha sido planificar los objetivos, el segundo paso consiste en desarrollarlos y el tercer paso en hacer los cambios pertinentes si se requieren. En esta fase se itera todas las veces necesarias para llevar a cabo el proyecto.
- En la fase 4, Beta, se tiene como objetivo evaluar y ajustar distintos aspectos del juego. Para ello, durante el proyecto se han hecho alguna versión Alpha y Beta, en la cual se han apuntado mejoras o errores a corregir.
- En la última fase, Cierre, tiene como objetivo entregar la versión final del videojuego al cliente (en este caso a la universidad) según las formas establecidas y evaluar el desarrollo del proyecto.

3. DESARROLLO DEL PROYECTO

En esta sección se hará una explicación a fondo del desarrollo del videojuego, el cual se ha estructurado a partir de la definición de tareas y subtareas en las cuales se ha iterado hasta darlas por finalizadas y dar paso a la siguiente tarea. Unity tiene un sistema de capas a la hora de poner los elementos en el espacio de desarrollo. A cada elemento se le asigna dicha capa para cuando este es mostrado en la ejecución del juego tenga la apariencia y funcionalidad que nosotros deseemos (por ejemplo: montañas o el cielo al fondo de la pantalla). Figura 2 muestra un ejemplo de las distintas capas que aparecerán en nuestro prototipo.

Figura 2: Capas definidas

Unity también posee una máscara de colisión la cual se implementa desde el entorno con el fin de crear el sistema colisionador que se quiera. La creación de esta máscara es crucial para el desarrollo del proyecto, ya que, por ejemplo, los proyectiles –tanto los disparados por el jugador como por los NPC- han de tener en cuenta con qué elemento pueden colisionar.

Finalmente hay que tener en cuenta de que para cada elemento de cada una de las capas ha sido necesaria la creación de prefabs. Un prefab es un elemento finalizado en el cual están los assets visibles por el usuario final, la relación entre éstos (en el caso de que este compuesto por varios, como el personaje principal) y los componentes asignados tales como scripts o elementos para el correcto funcionamiento.

Para facilitar la explicación de la metodología de programación se ha hecho una división según el elemento de juego que está involucrado (terreno, personaje controlado por el jugador, IA y resto de elementos).

3.1 Diseño de escenario y de los elementos del terreno

Para hacer el diseño del escenario ha sido necesario el uso de assets los cuales se pueden descargar desde la store de Unity, de otras webs o desarrollarlos con la ayuda de programas con la funcionalidad de pixel art [11].

Como se ha dicho anteriormente, se ha creado una parte inaccesible para el jugador en la que está el fondo, las montañas, rocas y árboles que decoran el escenario.

3.1.1 Elementos del terreno

Los elementos con los que puede interactuar el personaje principal son:

- Terreno por el cual el jugador puede moverse o saltar.
- Elementos no controlados por el jugador como cañones o IA.
- Algunos elementos que tienen funcionalidades como son plataforma movibles o con pinchos.

Es necesario recalcar que todos los elementos con los que interactuará el personaje principal han de tener el componente *ColliderBox2D*, el cual habilita la colisión entre elementos del juego.

3.1.2 Scripts necesarios para alcanzar funcionalidad deseada

Ha sido necesaria la creación de scripts para dar dinamismo al juego a algunos de los elementos mencionados para el diseño del escenario.

Para las trayectorias movibles se han generado dos scripts:

- *PathDefinition*: mediante una función indicamos el número de puntos que queremos que tenga la trayectoria y en la escena de trabajo de Unity se ven unidos todos los puntos por una línea dibujada por otra función.
- *FollowPath* permite dar inicio al movimiento mediante una función interna, y se va actualizando el movimiento según si hay punto al cual ir, y en caso de que sea el último punto, retroceder hasta el principio, para generar un movimiento infinito a partir de una trayectoria generada.

Cabe decir que mediante *FollowPath* se pueden definir dos tipos de velocidades para los movimientos de la plataforma: constante y con aceleración al salir de cada punto definido.

Figura 3 muestra una trayectoria con 3 puntos y la plataforma la cual seguirá el camino.

Figura 3: Ejemplo de trayectoria con 3 puntos de una plataforma

Además de plataformas con movimiento, se han definido otro tipo de plataformas que generan un impulso al jugador cuando éste salta encima. Para ello se ha programado un script adicional: *JumpPlatform* llama a una función interna, la cual teniendo en cuenta el valor de una variable que define la potencia de salto, genera un impulso de salto en cuanto el personaje principal entra en contacto con la plataforma la cual tenga asignado este script.

3.1.3. Creación de pantalla de juego y gestión de la cámara

Definición de la pantalla de juego

Una vez se tienen todos los elementos que queremos tener en el escenario y están listos para ser incluidos en la pantalla de juego, el siguiente paso consiste en dibujar la pantalla de juego e indicar los límites de la misma (que marcarán la longitud de pantalla del juego). En este caso los límites del escenario se han hecho mediante un *gameObject* [12] (Objeto al cual se le añadirán componentes para ponerlo en el espacio de unity) y un *BoxCollider* [13].

Asimismo y para facilitar la experiencia de juego en pantallas largas, se ha añadido la funcionalidad de checkpoints o puntos intermedios a los que vuelve el jugador una vez pierde una vida. Para implementar los checkpoints [14] en Unity, ha sido necesario crear un GameObject (Managers) en el cual generamos el numero de checkpoints que queramos siendo cada uno un nuevo GameObject.

Para obtener el funcionamiento deseado de los checkpoints han sido necesarios cuatro scripts: *IPlayerRespawnListener*, *Checkpoint*, *GameHud* y *LevelManager*.

- En el GameObject general que gestiona la pantalla entera (Managers) le añadimos como componente el script *LevelManager* el cual se ocupa de guardar en una lista todos los checkpoints disponibles. Este script calcula a qué checkpoint debe ir el jugador cuando muere mediante la posición del jugador en el eje X, siendo el último checkpoint pasado en el cual reaparecerá. También guarda el puntaje obtenido hasta ahora y en el caso de que el jugador muera antes de pasar cualquier checkpoint, éste reaparecerá en el Levelstart (primer gameObject de Managers).
- A cada checkpoint de Managers, se le ha asignado como componente el script *Checkpoint* el cual tiene una función que hace que cuando el jugador pase por el checkpoint, se guarde esa posición y se muestre por pantalla que se ha pasado por un checkpoint. En el caso de que el jugador muera pasado un checkpoint, este script hace reaparecer el jugador en el último checkpoint pasado.
- *GameHud* [15] es el encargado del funcionamiento del tiempo y puntaje mostrado en el lado superior izquierdo de la pantalla.
- *IPlayerRespawnListener* es una llamada a una función el cual se utiliza en el script *Checkpoint* para hacer reaparecer al jugador en el checkpoint.

Figura 4 muestra un ejemplo de diseño de terreno incluyendo decorados y checkpoints.

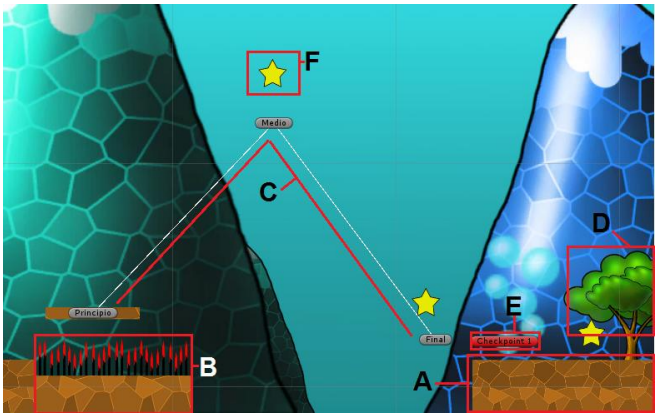


Figura 4: Ejemplo de pantalla

Tal y como se puede ver en la imagen, aparecen diferentes elementos en la pantalla: (a) terreno por el cual el jugador puede caminar; (b) terreno con pinchos; (c) la trayectoria de una plataforma movible; (d) elementos decorativos; (e) uno de los chekcpoints de la pantalla y (f) elementos como las estrellas para conseguir puntos.

Gestión de la cámara

Una vez tenemos definidos los límites del escenario (*Camerabounds*), el siguiente paso ha consistido en la programación de scripts para que la cámara que define la vista del juego por parte del usuario siga al personaje principal manteniéndolo en el medio de la pantalla.

Para ello generamos la vista que queremos que vea el jugador: Main Camera, y la ponemos del tamaño que se crea conveniente. Añadimos el script *CameraController* a Main Camera, donde se ha definido una variable llamada Player que una vez asignada al *gameObject* del jugador permite que la cámara siga al jugador. *CameraController* funciona de la siguiente manera: refresca lo que muestra por pantalla teniendo al jugador en el centro teniendo en cuenta los límites definidos por *CameraBounds*.

3.2 Diseño del personaje principal

Elementos y componentes de diseño

Para hacer el personaje principal, ha sido necesario crear un prefab a partir de un gameObject. En el gameObject se han añadido por separado cada una de las partes en que se ha dividido al personaje: cabeza, pies, ojos, arco, escudo etc. El personaje principal tiene: 2 componentes (BoxCollider2D, RigidBody2D) y tres scripts para implementar las funcionalidades del personaje: *Jugador*, *ControlesJugador2D* y *PlayerBounds*.

- En el script *Jugador* se declaran variables como la velocidad o la vida del personaje (Health) y funciones para generar el movimiento. Este script controla el uso de las teclas para mover al personaje, gestionando cuándo el personaje dispara y cambia el signo del eje según hacia donde mire el personaje. Asimismo controla aspectos relacionados con la variable vida del personaje: controla cuando la variable vida del jugador llega a 0, el restablecimiento del valor de la variable vida al 'renacer' en un checkpoint y variaciones en el valor de esta variable al recibir daño o recuperar vida.
- En *ControlesJugador2D* se controla todo el sistema de colisión [16] con el terreno generando en el *BoxCollider* un sistema para tener control sobre el contacto con otros elementos.
- En *PlayerBounds* se declara el tamaño de los límites de la pantalla, y se añaden acciones para cuando el personaje entre en contacto con estos límites.

Gestión de la vida del jugador

Para llevar a cabo la vida y el daño [17] que ésta pueda recibir y el contacto al impactar con las diferentes formas de IA (proyectiles y NPC's) han sido necesarios tres scripts diferentes: *HealthBar*, *FollowObject* y *GiveDamageToPlayer* y así como añadir funcionalidades en el script Jugador.

- En *HealthBar* se incluyen variables para tratar a nivel estético la barra de vida del personaje, desde el color de la barra con la vida llena hasta el color de cuando ha recibido daño. Este script es asignado al borde de la vida mediante un GameObject, el cual contendrá herramientas para visualización intuitiva del valor de la variable vida (barra azul) en otro GameObject.
- *FollowObject* es asignado a la barra de vida y le indicamos que siga al Jugador por pantalla en la posición que la hayamos puesto (encima del jugador).
- En *GiveDamageToPlayer* se tiene en cuenta la colisión del elemento que tendrá este script con el jugador, y en cuanto entren en contacto, al jugador se le restará la vida que pongamos como entrada en la variable DamageToGive.

Figura 5 muestra una imagen del personaje con la vida entera y al personaje con menos vida. Conforme vaya teniendo menos vida, el color irá pasando a rojo.

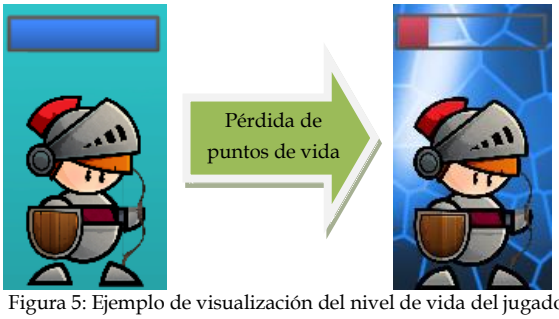


Figura 5: Ejemplo de visualización del nivel de vida del jugador

Diseño del personaje

Para el diseño del personaje se empezó usando un asset simple para las primeras pruebas pero se reemplazó por otro estéticamente más bonito el cuál ha sufrido otro cambio cambiando la espada por un arco ya que éste dispara y así asemejarse más a la acción real.

Figura 6 muestra la evolución del diseño del personaje:

Figura 6: Evolución de diseño del personaje principal

3.3 Diseño de personajes no controlados por el jugador

Para la IA [18] del juego ha sido necesario un script: *SimpleEnemyIA*. En este script se asocian los scripts de físicas del jugador (se le otorga una velocidad inicial y la capacidad de colisionar) a elementos controlados por la IA. Asimismo existe un control a la hora de colisionar con el terreno para que tenga una trayectoria infinita hacia una dirección y cambie de sentido en caso de colisionar.

Mediante la funcionalidad *RayCast* detectamos si el objeto que tiene la IA delante es Jugador, y en el caso de que así sea, emitirá un proyectil que quitará vida al jugador. Usando un mismo script se ha realizado una división de la complejidad de la IA en tres niveles:

- El primer nivel de complejidad consiste en una IA en movimiento. La colisión del jugador con la IA supone una modificación del valor de la variable Health.
- El segundo nivel implica una IA estática que es capaz de detectar la presencia del jugador. Asimismo la colisión con el jugador implica disminución del nivel de vida.
- El nivel más complejo junta ambos comportamientos, haciendo que la IA sea capaz de detectar al jugador, disparándole a la vez que tiene movimiento.

Figura 7 muestra el diseño de los enemigos del juego:

Figura 7: Ejemplo de diseño de personajes enemigos

3.4 Diseño del resto de elementos del juego

Sistema de puntuación del juego

A continuación se explicará el sistema de puntuación del juego, para ello han sido necesarios cuatro scripts: *GameManager*, *GameHud*, *PointStar* y *AutoDestroyParticleSystem*.

- *GameManager* contorla el puntaje mediante funciones que resetean el contador o añaden puntos a los ya existentes.
- *GameHud* nos muestra en la parte superior izquierda el tiempo de juego y el puntaje adquirido.
- *Pointstar* es el script que añadimos como componente a cada estrella (GameObject) en Unity. En este script se tiene en cuenta la futura colisión entre el Jugador y la estrella. Al producirse ésta, se genera un mensaje por pantalla indicando la cantidad de puntos recibida al alcanzar la estrella.
- Y por último, *AutoDestroyParticleSystem* oculta la aparición del objeto estrella, eliminando sus atributos destruyendo el GameObject asociado.

Visualización de variables con incidencia en el juego

Figura 8 muestra la representación en pantalla de los puntos adquiridos por el personaje, el tiempo transcurrido de juego en esa pantalla y una estrella la cual podrá coger el jugador.

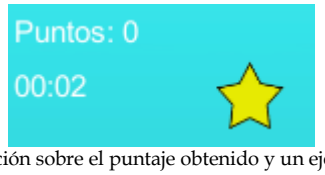


Figura 8: Información sobre el puntaje obtenido y un ejemplo de estrella

En el caso de que obtengamos alguna estrella, pasemos por algún checkpoint o perdamos vida mediante algún proyectil, se mostrará por pantalla los datos informativos necesarios [19]. Para ello han sido necesarios tres scripts: *FromWorldPointTextPositioner*, *IFloatingTextPositioner* y *FloatingText*.

- A partir de *FromWorldPointTextPositioner* obtenemos la posición de la pantalla la cual pasamos a *IFloatingTextPositioner* y ésta envía un bool a *FloatingText* con la información de la posición de la pantalla para poder mostrar el mensaje por pantalla.
- A *FloatingText* se le pasará el texto a mostrar, el estilo y la posición de pantalla para mostrarlo.

En Figura 9 se pueden ver ejemplos de texto por pantalla, en la primera imagen al recoger una estrella y los puntos que obtenemos y en la segunda imagen la información de que hemos pasado por un checkpoint.

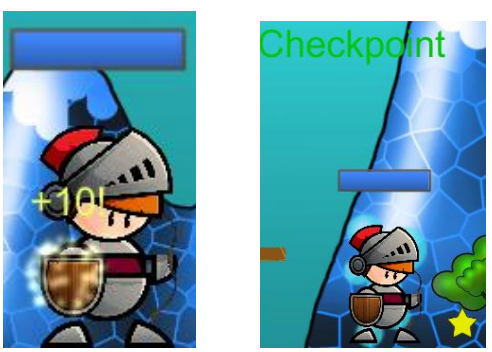


Figura 9: Ejemplos de texto por pantalla (izquierda) Incremento de puntuación al coger una estrella (derecha) Paso por checkpoint

Diseño y funcionamiento de proyectiles

En lo que respecta a los diferentes proyectiles [20] que se puede encontrar en el juego, hay diversos tipos: en el primer caso se explicará los proyectiles que salen del cañón, y para ello han sido necesarios dos scripts: *PathedProjectileSpawner* y *PathedProjectile*.

- Se le asigna a un GameObject *PathedProjectileSpawner* desde donde queremos que aparezca el proyectil, que estará dentro del GameObject Cañón. A este script se le pasa el GameObject el cual queramos de destino en la trayectoria del proyectil, el proyectil que queremos que dispare con sus correspondientes scripts añadidos (*PathedProjectile* y *GiveDamageToPlayer*), la velocidad a la que saldrá el proyectil y la velocidad de reaparición de nuevos proyectiles.
- *PathedProjectile* es un script que es asignado al proyectil que queramos que utilice el cañón. En este script se le indica el efecto que queremos que haga el proyectil al impactar en el GameObject Destino dentro del GameObject Cañón para tener un efecto de explosión, también se introduce el número de puntos que dará al jugador al destruir este tipo de proyectiles.

Figura 10 muestra el cañón disparando un proyectil con el efecto de explosión.

Figura 10: Ejemplo de cañón disparando

En el caso de los proyectiles que dispara el jugador y los NPC's se han implementado dos scripts: *Projectile* y *SimpleProjectile* y también se han añadido funcionalidades en Jugador.

- En *Projectile* tendremos la clase Proyectil para tener en cuenta las colisiones en las diferentes capas de elementos: en el caso del proyectil que disparará el jugador, éste podrá colisionar con las capas plataformas, proyectiles enemigos tanto del cañón como de los NPC's y con los propios NPC's para destruirlos. Esta clase se utilizará para el proyectil que disparará nuestro jugador y los NPC's.
- *SimpleProjectile* recoge la velocidad inicial del proyectil, las capas con las que podrá colisionar el proyectil definidas por la clase Proyectil del script Projectile, el daño que hará el proyectil, el efecto que hará al colisionar simulando una explosión y el tiempo que permanecerá el proyectil en el juego en caso de no colisionar con ningún elemento del terreno.
- Por último, en Jugador añadimos una función para que la persona que esté jugando al hacer click con el botón izquierdo del ratón, el personaje del juego dispare.

Para los efectos de explosión de los proyectiles se ha utilizado una función de Unity, *particle system* [21], el cual se puede modificar a gusto del desarrollador, estilo, forma, color, tiempo de duración etc.

Gestión de finalización de escena

Finalmente, para comprobar que se ha finalizado la escena actual y pasar a otra escena [22] o finalizar el juego, ha sido necesario un script *FinalStartPoint* asociado a un elemento del escenario (estrella grande en este caso). Al entrar en contacto el jugador principal con este elemento se activa el funcionamiento de *FinalStartPoint* el cual puede implicar que tras unos segundos se pasa a otra escena o se finaliza el juego.

3.5 INTERFAZ USUARIO/MÁQUINA

El sistema de control del personaje [23] es intuitivo y sencillo de utilizar basado en controles de juegos extendidos globalmente.

Mediante las teclas a y d controlamos al personaje para caminar hacia un lado u otro y mediante espacio se puede saltar. Respecto al manejo de los proyectiles, el personaje principal dispara al pulsar el botón izquierdo del ratón.

En cuanto al sistema de menús, el menú principal presenta un fondo con la temática del juego y un botón que lleva a la ejecución del mismo, una vez se termina el juego, aparece otra pantalla que indica que el juego ha terminado y otro botón para poder volver a jugar la partida.

Figura 11 y 12 muestra los controles que serán necesarios para manejar al personaje del juego:

Figura 11: Controles de juego

Figura 12: Controles de juego

4. RESULTADOS

En esta sección se detallarán los cambios producidos en la planificación inicial, algunos de los problemas encontrados durante el proyecto y el estado actual del juego.

4.1 CAMBIOS EN LA PLANIFICACIÓN

A lo que respecta a la planificación, todas las tareas propuestas inicialmente se han llevado a cabo. Los únicos cambios menores han consistido en posponer todas las tareas relacionadas con la interfaz de usuario (Menú principal y pantalla de final de juego) con el objetivo de adelantar trabajo de programación de IA. Asimismo se ha considerado necesario añadir otras tareas no propuestas inicialmente con el fin de ampliar las funcionalidades previstas. Las tareas añadidas han sido:

- Seguimiento de cámara y establecimiento de límites de pantalla.
- Introducción y manejo de checkpoints.
- Sistema de puntuación basado en estrellas.
- Manejo del control de la vida y daño del personaje.
- Introducción de texto flotante para comunicación máquina-jugador.
- Posibilitar el empleo de proyectiles
- Introducción de música y sonidos

4.2 PROBLEMAS ENCONTRADOS/BUGS

A principios del proyecto, se encontraron pequeños problemas achacables a empezar a trabajar con un entorno completamente nuevo (Unity). Estos problemas fueron solucionados tras un periodo de aprendizaje del mismo.
Los problemas más destacados han venido por problemas derivados de la herramienta en sí, que presenta algún tipo de inconsistencia respecto a las colisiones entre elementos del juego. Más concretamente, al añadir la posibilidad de colisionar (*BoxCollider2D*) a los elementos del terreno, si la creación del prefab asociado no ha sido en la posición espacial (x,y,z)=(0,0,0), al contactar el personaje principal con el perfil izquierdo del terreno al saltar, los BoxCollider entran en conflicto impidiendo el movimiento libre del personaje haciendo imposible el avance en el juego.

Por último y respecto al movimiento del personaje, se tuvo que considerar que el personaje ha de ser creado en la posición del espacio de Unity (x,y,z) = (0,0,0). Si esto no se hace así, el movimiento del personaje no será el esperado, haciendo que la posición del gameObject (personaje entero) no cambiase cuando el personaje sí se había trasladado de posición en pantalla.

4.3. ESTADO ACTUAL DEL PROYECTO

El estado actual de desarrollo es fase **beta**. El videojuego es completamente funcional y jugable. Actualmente consta de dos pantallas diferentes incluyendo funcionalidades de inicio de juego, salto de pantalla y música.

5. CONCLUSIONES

5.1 OBJETIVOS DE APRENDIZAJE ADQUIRIDOS

El objetivo principal de este trabajo fue el poder ser capaz de diseñar e implementar desde cero un videojuego de plataformas 2D. Asimismo, la realización del proyecto ha permitido introducirse en el desarrollo de videojuegos obteniéndose resultados satisfactorios. El proyecto ha sido completado con todos los objetivos iniciales cubiertos e incluso con varios objetivos puestos a mitad de proyecto para aumentar las funcionalidades del juego.

El manejo de la herramienta Unity ha confirmado la utilidad de la misma, ya que es capaz de relacionar elementos visuales con scripting de manera muy intuitiva, permitiendo obtener resultados satisfactorios.

Una vez realizado este trabajo y tras el proceso de aprendizaje, es posible poder realizar trabajos futuros más complejos en Unity una vez adquirida la base sólida y teniendo en cuenta la documentación disponible, al ser un motor gráfico muy extendido y utilizado a día de hoy.

5.2 LINEAS FUTURAS DE TRABAJO

Como líneas futuras, se enfocaría el trabajo realizado a un desarrollo más enfocado a la inteligencia artificial, ya que prácticamente se han cubierto todos los elementos que puede tener un videojuego plataformero.

Respecto a posibles ampliaciones del juego actual, el próximo paso podría consistir en crear un jefe final de juego el cual pudiese leer los movimientos del jugador y actuase en consecuencia, para que tuviese una dificultad añadida el pasarse el nivel del juego.

Otro de los campos en los se desearía profundizar sería en la mejora de la interfície gráfica, ya que dado el tiempo dado y que el objetivo era hacer el videojuego, no se ha podido hacer hincapié en este aspecto.

También relacionado con el anterior punto, sería interesante desarrollar una historia e implementar mediante pixel art los personajes y el terreno a utilizar para dar mas originalidad y personalidad al juego, ya que los utilizados en este proyecto han sido en su gran mayoría imágenes descargadas o simplemente retocadas con GIMP.

AGRADECIMIENTOS

Me gustaría dar las gracias a mi tutor Jorge Bernal por tutorizar este proyecto y hacer posible mi deseo de desarrollar un videojuego como proyecto final de grado y también por toda la ayuda recibida durante el proceso, por todas las reuniones y los correos que hemos intercambiado.

BIBLIOGRAFÍA

[1] Estudio de mercado sobre el mundo del videjuego
http://tinyurl.com/l83jwap
Última consulta: 24/10/2015

[2] Metodología SUM
http://tinyurl.com/q6u2odh
Última consulta: 03/11/2015

[3] Página web general de Unity
https://unity3d.com/es
Última consulta: 06/02/2015.

[4] Funcionamiento de scripts en C#
http://tinyurl.com/ptbz88f
Última consulta: 20/01/2015
[5] Web con el soporte digital e información sobre GIMP
http://www.gimp.org.es/.
Última consulta: 02/10/2014.

[6] Historia de los videojuegos
http://tinyurl.com/6fzb9f
Última consulta: 02/10/2014.

[7] Entorno de desarrollo de videojuegos Shiva 3D
http://tinyurl.com/m7zjyc7
Última consulta: 24/10/2014

[8] Entorno de desarrollo de videojuegos GameMaker Studio
http://tinyurl.com/mz3jvh6
Última consulta: 24/10/2014

[9] Entorno de desarrollo de videojuegos Torque 2D
http://tinyurl.com/qy9mrd4
Última consulta: 24/10/2014

[10] Tienda de Unity
http://tinyurl.com/klbztns
Última consulta: 27/12/2014

[11] Web de información de Pixel Art
http://tinyurl.com/olpmxm4.
Última consulta: 06/02/2015.

[12] Información sobre el uso de GameObjects en Unity.
http://tinyurl.com/q24hz9a
Última consulta: 15/01/2015

[13] Información sobre BoxCollider2D en Unity.
http://tinyurl.com/q2guyjo
Última consulta: 31/01/2015

[14] Información sobre el control de la cámara y check-points.
http://tinyurl.com/kkkucmq
Última consulta: 22/12/2014

[15] Información sobre creación de HUD y menús en videojuegos.
http://tinyurl.com/o62aaek
Última consulta: 25/12/2014

[16] Información sobre colisiones de elementos en video-juegos Unity.
http://tinyurl.com/n7l5az2
Última consulta: 20/12/2014

[17] Manejo de estado de variables de personaje.
http://tinyurl.com/ngd9j6w
Última consulta: 08/01/2015

[18] Inteligencia Artificial en Unity
http://tinyurl.com/nfmj7ww
Última consulta: 24/01/2015
[19] Información sobre texto en pantalla.
http://tinyurl.com/lzs5e2u
Última consulta: 05/01/2015

[20] Información sobre colisión de proyectiles
http://tinyurl.com/pp5b54b
Última consulta: 13/01/2015

[21] Información sobre particle system
http://tinyurl.com/nen3aol
Última consulta: 16/01/2015

[22] Definición de límites de pantalla y paso a escena diferente
http://tinyurl.com/k5oy6m8
Última consulta: 25/01/2015

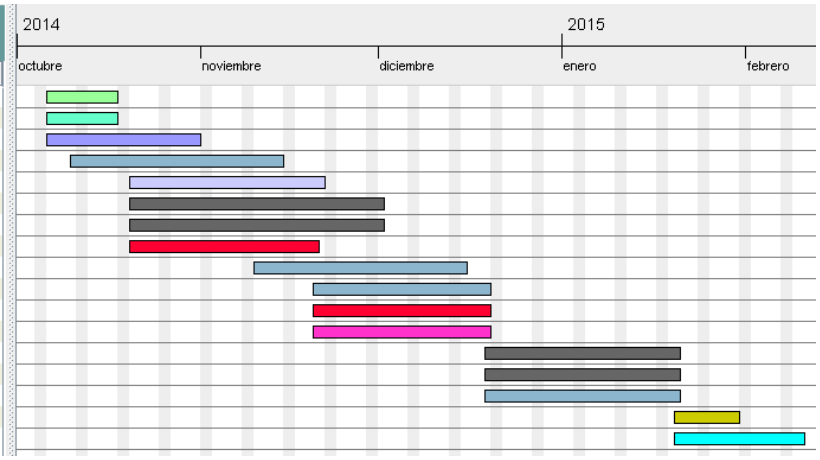
[23] Manual de usuario para Unity
http://tinyurl.com/nmcmk3g
Última consulta: 27/01/2015

APÉNDICES

1. DIAGRAMA DE GANTT DEL PROYECTO

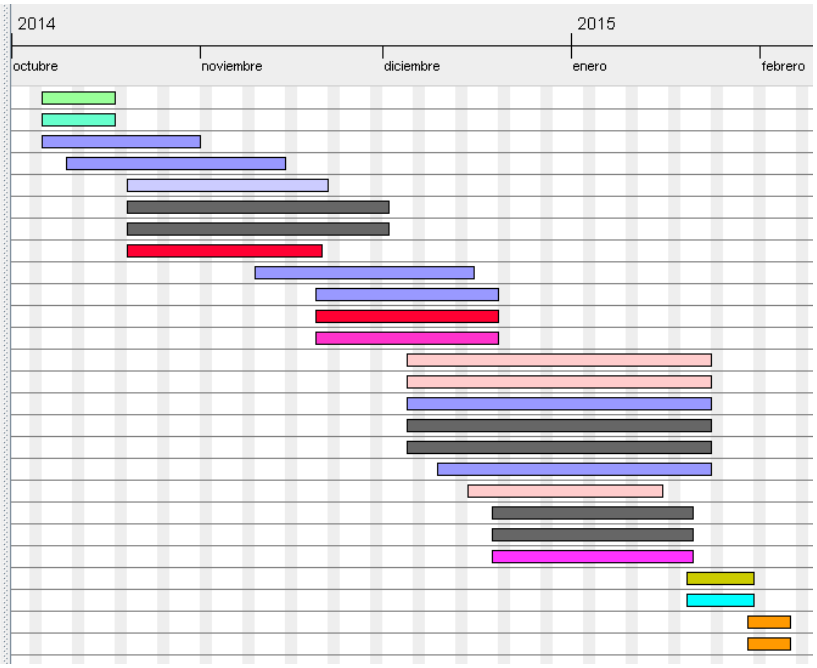
A continuación se muestra el diagrama de Gantt inical y el diagrama de Gantt final tras añadir tareas nuevas:

Nombre	Fecha de inicio	Fecha de fin
Captura de requisitos	6/10/14	17/10/14
Toma de contacto con Unity	6/10/14	17/10/14
Diseño UI	6/10/14	31/10/14
Implementacion inicial UI	10/10/14	14/11/14
Desarrollo del terreno unity	20/10/14	21/11/14
Diseño personaje principal	20/10/14	1/12/14
introduccion del pj principal	20/10/14	1/12/14
interaccion de pj con entorno	20/10/14	20/11/14
Funcionalidad menu	10/11/14	15/12/14
Diseño e implementación UI Fase2	20/11/14	19/12/14
interacción de pj con entorno fase2	20/11/14	19/12/14
inteligencia artificial	20/11/14	19/12/14
introduccion npc's	19/12/14	20/01/15
Diseño npc's	19/12/14	20/01/15
inteligencia artificial fase2	19/12/14	20/01/15
musica	20/01/15	30/01/15
IA fase3, retoques al diseño, mas ...	20/01/15	10/02/15



PLANIFICACIÓN INICIAL

Nombre	Fecha de inicio	Fecha de fin
Captura de requisitos	6/10/14	17/10/14
Toma de contacto con Unity	6/10/14	17/10/14
Diseño UI	6/10/14	31/10/14
Implementacion inicial UI	10/10/14	14/11/14
Desarrollo del terreno unity	20/10/14	21/11/14
Diseño personaje principal	20/10/14	1/12/14
Introduccion del pj principal	20/10/14	1/12/14
Interaccion de pj con entorno	20/10/14	20/11/14
Funcionalidad menu	10/11/14	15/12/14
Diseño e implementación UI Fase2	20/11/14	19/12/14
Interacción de pj con entorno fase2	20/11/14	19/12/14
Inteligencia artificial	20/11/14	19/12/14
Seguimiento de la camara	5/12/14	23/01/15
Introducción de checkpoints	5/12/14	23/01/15
Sistema de puntuación	5/12/14	23/01/15
Control de daño y vida del pj	5/12/14	23/01/15
Proyectiles	5/12/14	23/01/15
Texto emergente	10/12/14	23/01/15
Definición de límites de pantalla	15/12/14	15/01/15
introduccion npc's	19/12/14	20/01/15
Diseño npc's	19/12/14	20/01/15
Inteligencia artificial fase2	19/12/14	20/01/15
Musica	20/01/15	30/01/15
IA fase3, retoques al diseño, mas ...	20/01/15	30/01/15
Preparar documentos	30/01/15	5/02/15
Testeo fase beta	30/01/15	5/02/15



PLANIFICACIÓN FINAL

2. EJEMPLO DE UNA PANTALLA CON LOS ELEMENTOS DEL JUEGO