MEMORIA TFG

Especialidad en Desarrollo de Aplicaciones Interactivas (M2)

CREACIÓN DE UN VIDEOJUEGO 2D CON UNITY
PLATAFORMAS - PUZLE

UOC
Universitat Oberta de Catalunya

Roberto Alcázar Alcázar

Consultor: **Kenneth Capseta Nieto**
Profesor: **Carlos Casado Martínez**

16 Enero 2017

COPYRIGHT

· Esta MEMORIA esta sujeta a las siguientes licencias:

creative commons

BY — Debe reconocer adecuadamente la autoría, proporcionar un enlace a la licencia e indicar si se han realizado cambios. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador o lo recibe por el uso que hace.

NC — No puede utilizar el material para una finalidad comercial.

ND — Si remezcla, transforma o crea a partir del material, no puede difundir el material modificado.

· El VIDEOJUEGO esta sujeto a la siguientes licencias:

DEDICATORIA / CITA

“Dedicado a mis padres, hermanos, amigos, compañeros de trabajo y a todas esas personas que de una forma u otra han pasado por mi vida durante esta larga travesía. Perdón por el tiempo que no os he dedicado y gracias por vuestra comprensión, paciencia y ayuda.”

“Está vivo. Está vivo”
Frankenstein (1931)

“Welcome to the real world”
Morpheus - The Matrix(1999)

ABSTRACT

El objetivo de este proyecto es la creación de un **videojuego 2D** de **plataformas** y **puzzles** en todas sus etapas, desde la **conceptualización**, creación de un **guion**, **diseño** y **programación** final del juego.
La elección de este tipo de proyecto tiene dos motivaciones principales:

Por un lado un videojuego es que un tipo de producto que aúna **múltiples disciplinas** tanto **técnicas** (programación, BD, mecánicas, física…) como **artísticas** (diseño, animación, ilustración…) sin olvidar el apartado de conceptualización, guion, documentación… apartado audiovisual, etc. Con todos estos procesos necesarios para producir un videojuego abarco de forma completa la formación recibida en el **Grado Multimedia**.

Por otro lado la elección de **Unity 5.x** como herramienta de desarrollo y **C#** como lenguaje de programación, es algo que he echado en falta en el temario optativo del Grado y por tanto no podía desaprovechar la oportunidad para aprenderlos. Pienso que conocer Unity me puede ser muy útil a nivel profesional por su potencia y versatilidad para crear proyectos **multiplataforma**.

Para finalizar debo aclarar los objetivos finales propuestos; dado que el tiempo de desarrollo del proyecto no es todo extenso que requeriría la producción de un producto final acabado de cierta calidad, he decidido limitar la ejecución a una demo funcional jugable con sólo algunos personajes, monstruos, mecánicas y escenarios, sin que esto delimite la parte inicial creativa de guion y trasfondo, que me permita **continuar el desarrollo** una vez concluida la entrega del TFG.

El juego correrá como aplicación sobre **Windows** y sobre **navegador web** con plug-in de Unity y requerirá de teclado para controlarlo. Una versión para dispositivos móviles se contemplaría como una segunda etapa tras la finalización del proyecto.

Palabras clave: videojuego, Unity, programación, C#, diseño, Photoshop, Ilustrator, Indesign, conceptualización, arte, mecánicas, ilustraciones, efectos, música, interfaz, animación, plataformas, puzle.

NOTACIONES

A continuación se describen las diferentes notaciones que contiene este documento.

Título Sección: Tipografía Calibri 22pt RGB (255, 255, 255)

ejemplo:”NOTACIONES ”

Texto interior: Tipografía Calibri 14pt RGB (27, 102, 143)

ejemplo: “la creación de un**”**

Texto interior destacado: Tipografía Calibri 14pt Bold RGB (27, 102, 143)

ejemplo: “**videojuego 2D** de **plataformas”**

Citas: Tipografía Calibri 18pt Italic y entre comillas “” RGB (27, 102, 143)

ejemplo: *“Welcome to the real world”*

Código fuente: Calibri 10pt RGB (27, 102, 143)

ejemplo:

```
using UnityEngine;
using System.Collections;
using System;
using UnityEngine.SceneManagement;
```

AGRADECIMIENTOS

Muchas gracias a todos los que me habéis ayudado en la elaboración de este juego, con vuestros consejos, ideas, infinidad de testeos… y sobretodo críticas constructivas. Sin vuestra aportación no creo que hubiera tenido una experiencia y resultado tan digno.

Gracias.

INDICE

1 INTRODUCCIÓN

La **industria del videojuego** no ha parado de crecer hasta convertirse en un producto de consumo de masas. Desde la aparición de los primeros videojuegos modernos en la década de los 60, hasta la actualidad, hemos asistido a una continua transformación del medio pasando de ser producciones individuales o de pequeños equipos multidisciplinares a producciones millonarias donde cientos de personas especialistas trabajan de forma conjunta para la realización de la superproducción.

El género de **plataformas** es uno de los primeros en aparecer y se hizo muy popular en la década de los 80 y 90 y, sin duda, formó gran parte del entretenimiento de mi infancia y por este motivo le tengo un cariño especial.

Actualmente, en un mercado dominado por las grandes producciones, es un género con un volumen minoritario pero que tiene su público que busca una experiencia más "casual" ya que, en general, este tipo de juegos permite una dedicación más esporádica y sin grandes complicaciones en el aprendizaje de las mecánicas.

Figura 1 - SUPER MARIO BROS

Figura 2 - UNRAVEL

Un videojuego no es solo un medio de entretenimiento, es también un medio para comunicar, pensar, imaginar...

2 DESCRIPCIÓN

El proyecto consiste en la creación de un videojuego *indie* para Windows y que alternativamente podrá ejecutarse a través del navegador web gracias al plug-in de Unity.

Dentro de los géneros de videojuego existentes se podría clasificar como de **acción** y **plataformas**, aunque me gustaría incluir algunos elementos tipo **puzle** para proporcionar al usuario un reto mayor.

El juego narrará la historia de **un pequeño dragón** que tendrá que encontrar a sus otros hermanos ocultos por un escenario donde tendrá que superar trampas y vencer a enemigos. A medida que rescate a otros dragones estos podrán ser utilizados por el usuario, reemplazando al personaje principal, para poder superar ciertos puntos, ya que cada dragon contará con habilidades diferentes.

MEGAMAN 3 · Nintendo NES (1990)

Figura 3 - Megaman 3

UFOURIA · Nintendo NES (1991)

Figura 4 - UFOURIA

Sagas como **Megaman** o juegos como **Ufouria** serán una referencia en cuanto a mecánicas; estos juegos huyen en cierta forma de la linealidad común en los juegos de plataformas y dan al usuario cierta libertad en la toma de decisiones. El primero dispone de un orden de pantallas no lineal y en el segundo hay libertad de movimiento por el mundo donde discurre la aventura. Estos mecanismos siempre me han parecido interesantes ya que provoca que el usuario deba pensar para tomar las decisiones más adecuadas.

3 OBJETIVOS

Es necesario establecer unos objetivos que sirvan como hoja de ruta de trabajo para la consecución con éxito del proyecto. Dado que ésta es una aventura nueva, con nuevas herramientas de desarrollo y procesos que por ahora me son desconocidos, hay que establecer unos objetivos principales básicos y otros adicionales.

De esta forma se pueden concentrar los esfuerzos en completar las tareas fundamentales y, en caso de disponer de más tiempo, algunas adicionales. Todo en función de las dificultades que surjan y el tiempo disponible.

3.1 PRINCIPALES

· Realización de una demo jugable de un juego de plataformas 2D con Unreal Engine 5.x
· Crear un guion, diseño y programación escalables que permitan la ampliación, mejora y continuación del proyecto una vez entregado el TFG.
· El producto tendrá una estética y acabado visual agradable.
· Creación, animación e implementación de mecánicas para al menos 1 personaje principal y 2 secundarios.
· Creación de un escenario con recursos visuales variado y que permita una duración de juego de entre 5 y 10 minutos.
· Poner en práctica los conocimientos adquiridos en el Grado Multimedia en las áreas de producción, diseño y programación.
· Adquirir experiencia y nuevos conocimientos incorporando herramientas como Unity, lenguajes como C# y procesos mediante el uso de otras herramientas de animación.

3.2 SECUNDARIOS

· Crear una movie de introducción.
· Crear una base de datos con puntuaciones y conectividad con redes sociales.

4 MARCO TEÓRICO / ESCENARIO

Los **videojuegos independientes** son desarrollados por personas o compañías pequeñas de forma independiente y cuentan con presupuestos pequeños lo que obliga a una distribución digital de las copias. Aunque la falta de recursos provoca que el apartado gráfico tenga un acabado más sencillo que los juegos desarrollados por los grandes de la industria, su potencial reside en que los desarrolladores no tienen limitaciones o imposiciones de las editoriales, lo que lleva a obras más creativas.

La industria de los videojuegos ha despertado un creciente interés por los juegos independientes sobre todo desde la llegada de **internet** a la mayoría de hogares, convirtiendo a las conexiones de banda ancha en el canal ideal para la **distribución**. La comunidad indie se concentra entorno a varios portales de distribución como **Steam** o **Xbox live arcade** que actúan como intermediarios entre desarrolladores y usuarios finales. Estas plataformas concentran un gran catalogo donde se organizan los juegos en función de su temática posibilitando las búsquedas de juegos similares, filtrados por precio, etc. Este servicio no está exento de coste para el desarrollador, que ve como de cada venta una parte se la queda el intermediario. Pese a esto sigue siendo un canal de distribución más adecuado para el desarrollador indie que el soporte físico. Actualmente estas plataformas cuentan con miles de títulos disponibles y en aumento.

Este auge de videojuegos indie también ha venido propiciado por un **aumento** y simplificación de las **herramientas** para desarrolladores que simplifica enormemente la tarea en comparación con épocas pasadas. Por otro lado el aumento de **dispositivos móviles** inteligentes, como smartphones y tabletas, capaces de correr juegos ha ampliado el número de lugares y situaciones donde un usuario puede disfrutar de un videojuego.

La industria del videojuego supone cerca de 100.000 millones de dólares anuales y sigue en aumento.

5 CONTENIDOS

El videojuego dispondrá de:

· Menú principal de elección de opciones: Jugar, Controles, Creditos, Salir.

· Un nivel donde el usuario controlará un personaje y podrá realizar las siguientes ac ciones: moverse a izquierda y/o derecha, saltar sobre plataformas, eliminar enemigos saltando sobre ellos, utilizar su habilidad especial.

· Un evento de jefe final donde el jugador se enfrentará contra un enemigo que dispon- drá de sus propias mecánicas.

· En todo el juego habrá eventos de colisiones y físicas.

· Punto de control en el jefe final.

· Contadores de puntos de salud y habilidad mágica.

· Música y efectos de sonido.

6 METODOLOGÍA

Para poder desarrollar este videojuego hay que avanzar en diferentes áreas disciplinares.
La primera de ella es la elaboración de la idea principal, lo que podríamos llamar el **GUIÓN** de la obra. Aquí se define el título, historia, contexto, hilo argumental, características de los escenarios, personajes, monstruos, mecánicas, menús… Es importante definir todo lo máximo posible para poder planificar las tareas de producción de forma eficaz.
El software que se utilizará en esta etapa será **Mockups** para la creación de esquemas de pantallas y **MS Word** para el guion.

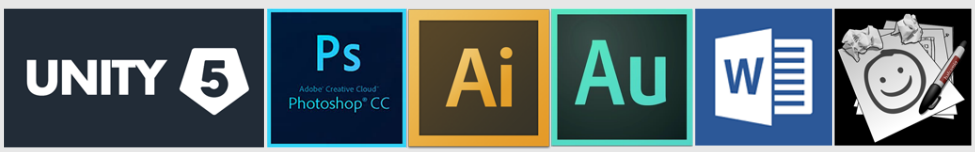
Una vez definido el proyecto pasamos al desarrollo del mismo que correrá a cargo del motor de videojuegos multiplataforma **Unity versión 5.x**. Esta herramienta permite la creación de juegos en 2D y 3D. En este caso se ha optado por un tipo de **proyecto 2D**.
El lenguaje de **programación** elegido será **C#** por su potencia y uso en el mercado. Con él, crearemos los scripts que se encargarán de controlar la física, movimiento de objetos en pantalla, eventos… en definitiva, la programación del videojuego. En la primera fase se utilizarán gráficos temporales que posteriormente serán substituidos por los definitivos, aunque esta etapa se superpone con la de diseño a medida que sea necesario ajustar los diseños a las mecánicas.

El **diseño** tendrá también diferentes etapas, empezando por bocetos creados “a mano alzada” hasta su digitalización mediante herramientas como **Photoshop** e **Ilustrator**. En este punto se crearán los gráficos de menús, pantallas y elementos como personajes, enemigos, etc.

Una vez creados los diseños entramos en fase de **animación**. Las animaciones en pantalla serán producto de la programación y de las animaciones creadas mediante **Unity** o aplicaciones como **Spine** o **Spriter**. Estas animaciones tendrán que ajustarse a las mecánicas implementadas, como saltos del personaje, estado idle, etc.

Por último la edición de las **músicas** y **efectos de sonido** deberán ser tratados y editados de ser necesario en **Adobe Audition** o similar.

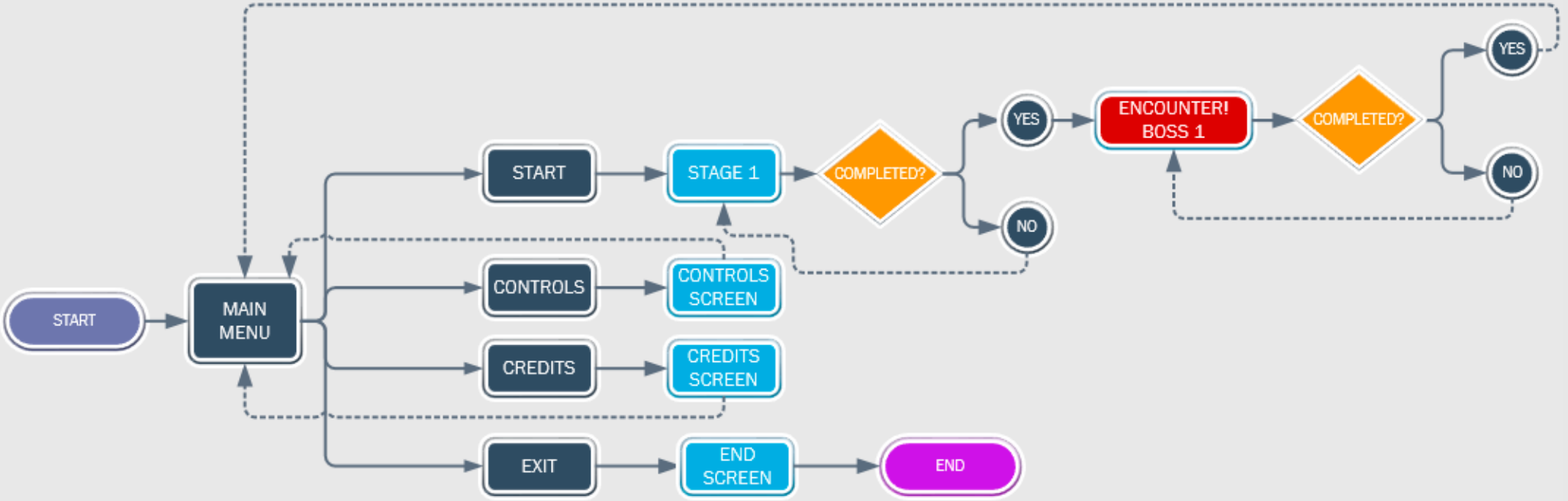
El desarrollo se irá documentando en esta **memoria** en el transcurso del avance del proyecto teniendo en cuenta las entregas.



9 ARQUITECTURA DE LA APLICACIÓN

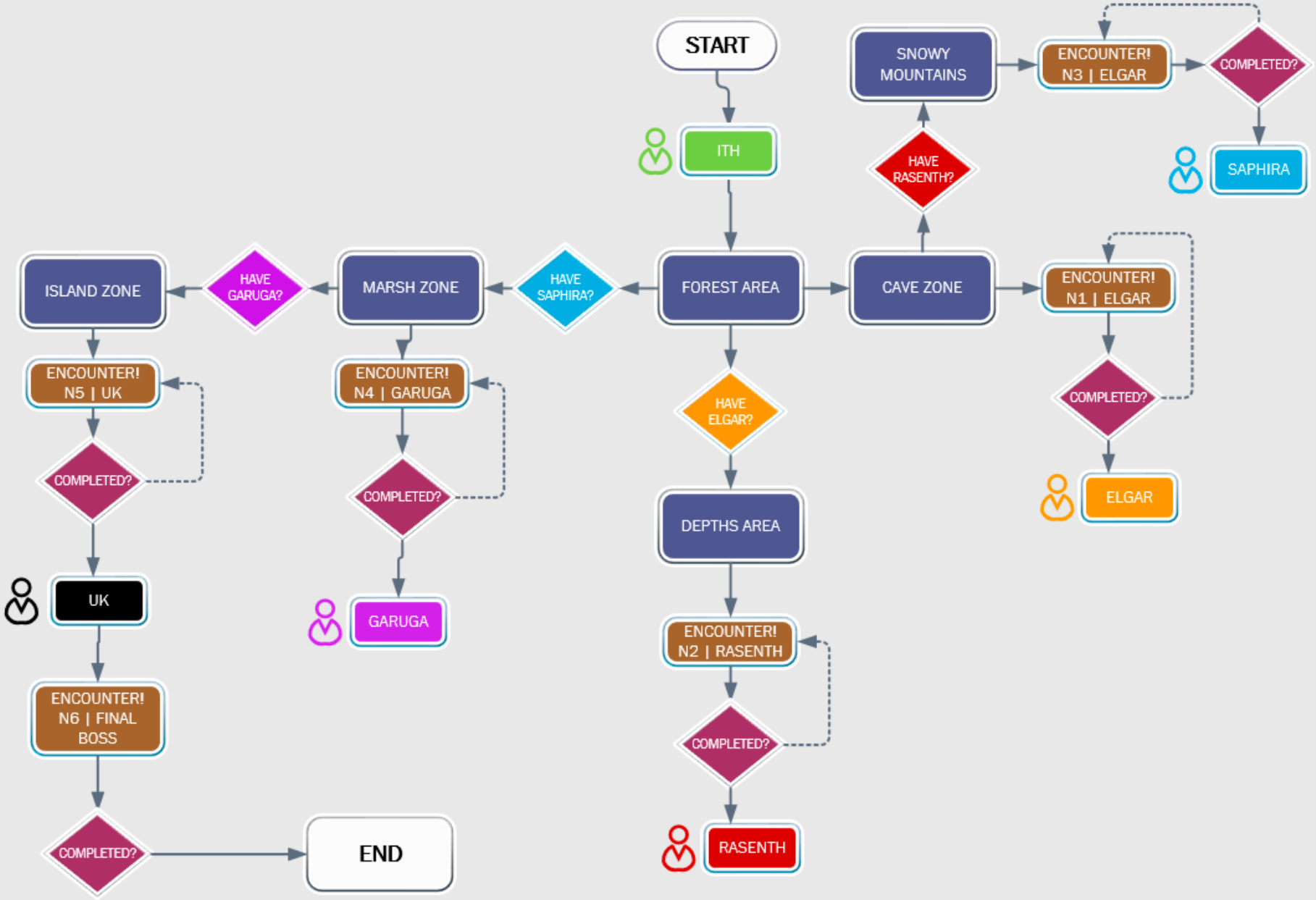
Los siguientes **diagramas de flujo** muestran:

· El primer diagrama refleja la **arquitectura** de la aplicación en cuanto a las opciones de **navegación** que encontrará el usuario.



· Este diagrama muestra la **arquitectura narrativa** de la **demo** que se pretende realizar para la entrega final.

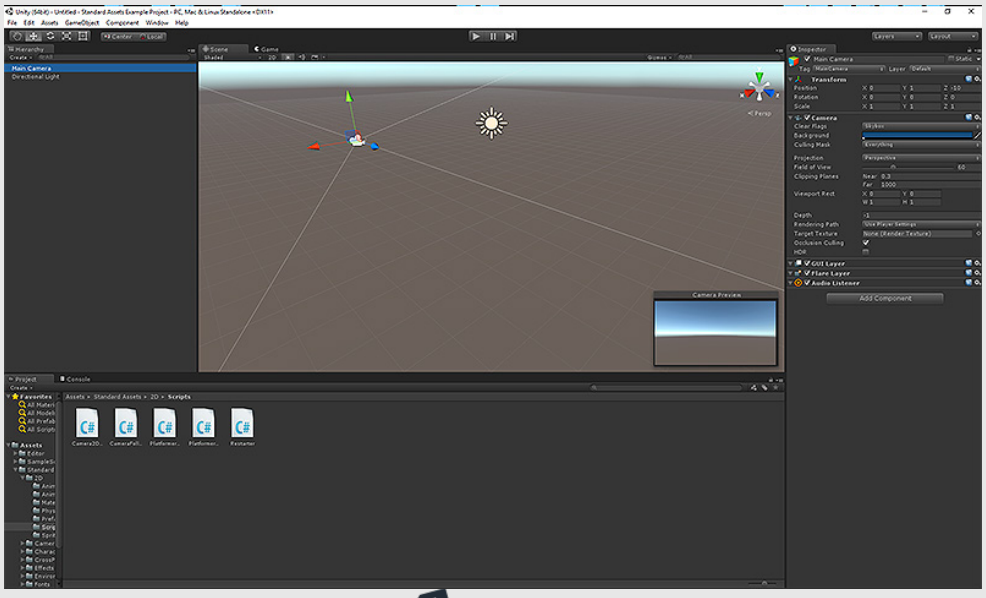
· Por último, el 3er diagrama muestra la **arquitectura narrativa del juego completo**, tarea a continuar después de la entrega final.



PLATAFORMA DE DESARROLLO

El desarrollo se realizará a través del motor de videojuegos multiplataforma **Unity** en su versión **5.X**. Las ventajas de utilizar esta herramienta son:

- Capacidad para compilar para diferentes tipos de plataforma. Motor gráfico Direct3D (Windows), OpenGL (Mac y Linux), OpenGL ES (Android e iOS), WebGL (web).
- Incorpora prácticamente todos los elementos necesarios para el desarrollo: Motor de físicas para detectar colisiones, animación de sprites, audio, generación de partículas, construcción de escenarios, programación, etc.
- Lenguaje de programación C#.
- Soporte de Photoshop para la edición de sprites.



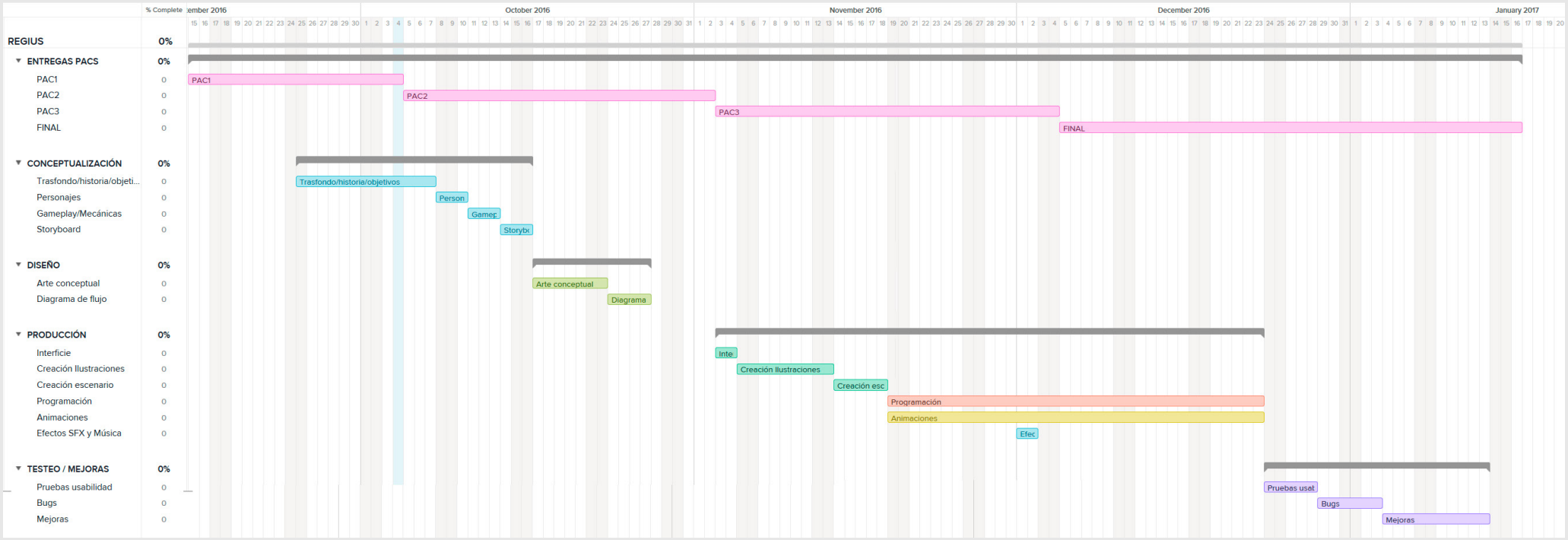
unity

9 PLANIFICACIÓN

La planificación se divide en varias etapas: **conceptualización**, **diseño**, **producción** y **testeo**. Además durante todo el proyecto se complementa esta **memoria**.

Aunque las áreas quedan definidas cronológicamente, dado que tanto Unity como C# y las herramientas de animación son desconocidas para mi hay una fase de aprendizaje de estas herramientas que abarca todo el proceso de principio a fin.

En paralelo a la producción iré cumplimentando la memoria periódicamente y no dejando esta tarea para los días finales a cada entrega.



10 PROCESOS DE TRABAJO

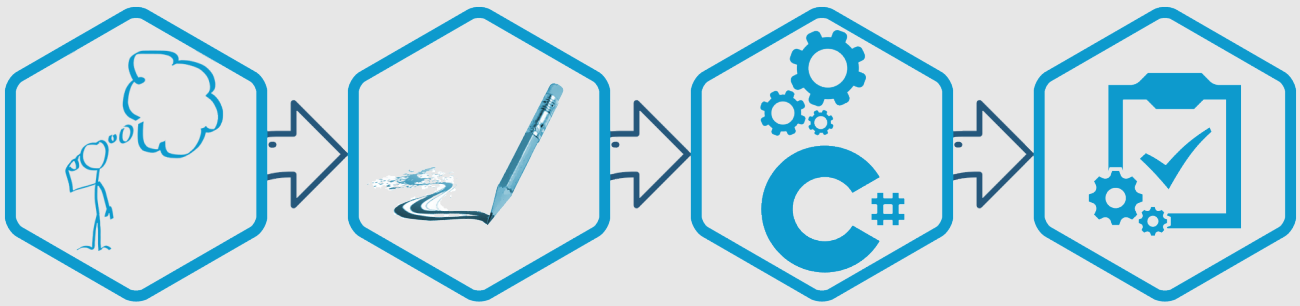
La producción se desarrolla en diferentes etapas:

· Conceptualización/Aprendizaje
Es la etapa inicial donde se desarrolla el argumento, personajes, ambientación… del juego. A través de un proceso de brainstorming se toman y desechan ideas hasta configurar un puzle atractivo.
En esta etapa se realizan esbozos de escenarios, personajes así como se definen las mecánicas a desarrollar posteriormente.
En paralelo se realiza una tarea de recopilación de documentación, avanzamos en el aprendizaje del lenguaje C# y hacemos una primera toma de contacto con la herramienta Unity.

· Diseño
Una vez tenemos definido como va a ser nuestro juego es el momento de traducir los esbozos creados de personajes y elementos del escenario a gráficos que serán utilizados en nuestro juego. Creamos los sprites necesarios a través de Ilustrator y Photoshop.

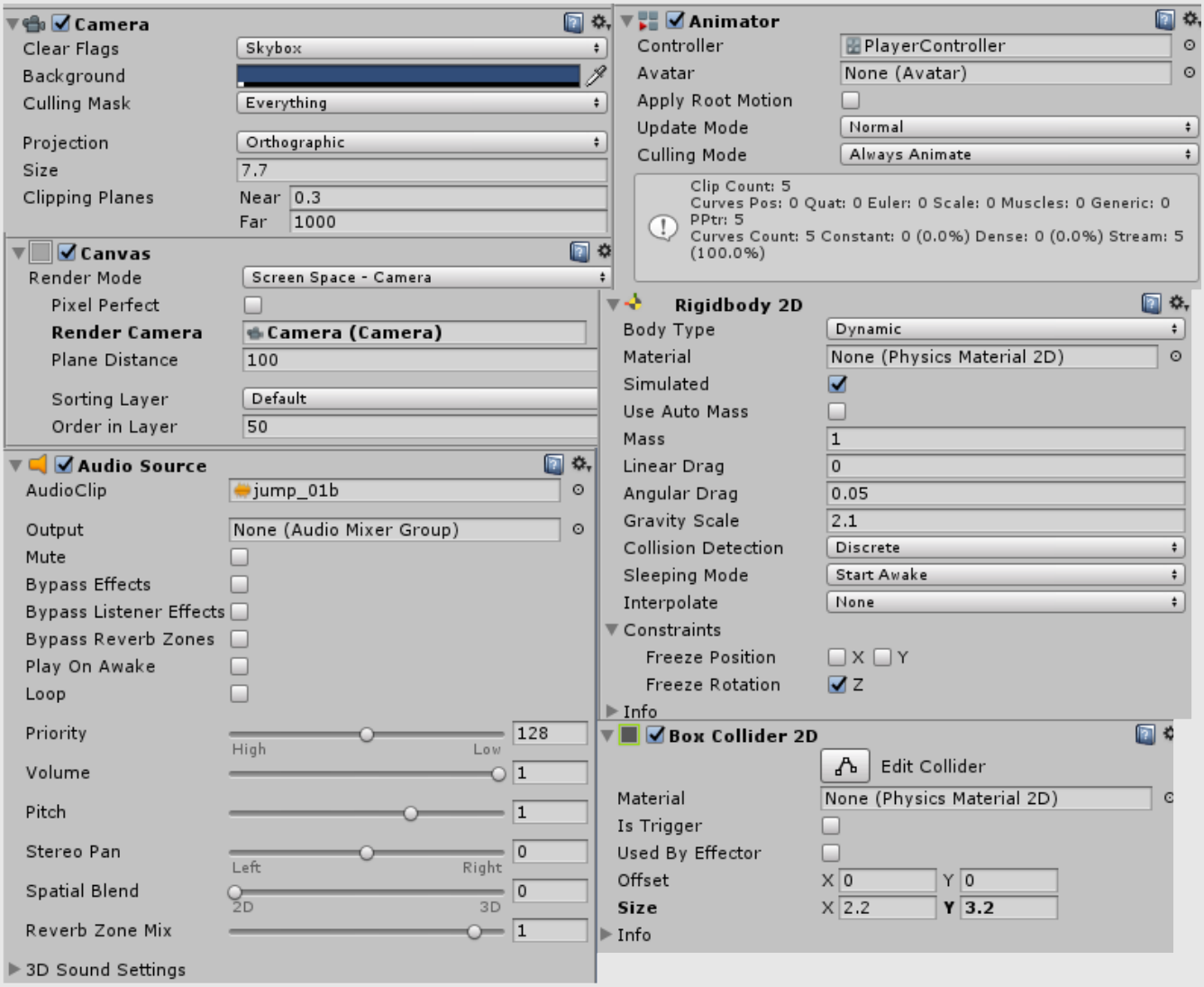
· Producción
Es la fase principal donde se realiza el ensamblado de los gráficos creados previamente, se crean las animaciones, efectos de sonido y programación.

· Testeo
Es la etapa final donde hacemos las últimas modificaciones en base a la experiencia que el juego transmite y gracias al feedback que puedan dar otros usuarios.



11 APIS UTILIZADAS

Estos son algunos de los componentes que UNITY permite que asociemos a los GameObjects y que he utilizado en el juego:



El componente **Rigidbody2D** permite dotar de físicas a un GameObject, en el podemos configurar la masa, gravedad, fijar algún eje de coordenadas, etc. Los objetos con gravedad 1 automáticamente caerán al reproducir el juego y por tanto requieren del uso de Colliders para que colisionen.

Los tipos de objetos **canvas** se utilizan para confección la UI del juego, portrait, vida del personaje, contador de estrellas y piezas, menús del juego… permiten que a diferentes resoluciones estos se escalen de forma no proporcional con el juego para permitir que siempre sean visibles.

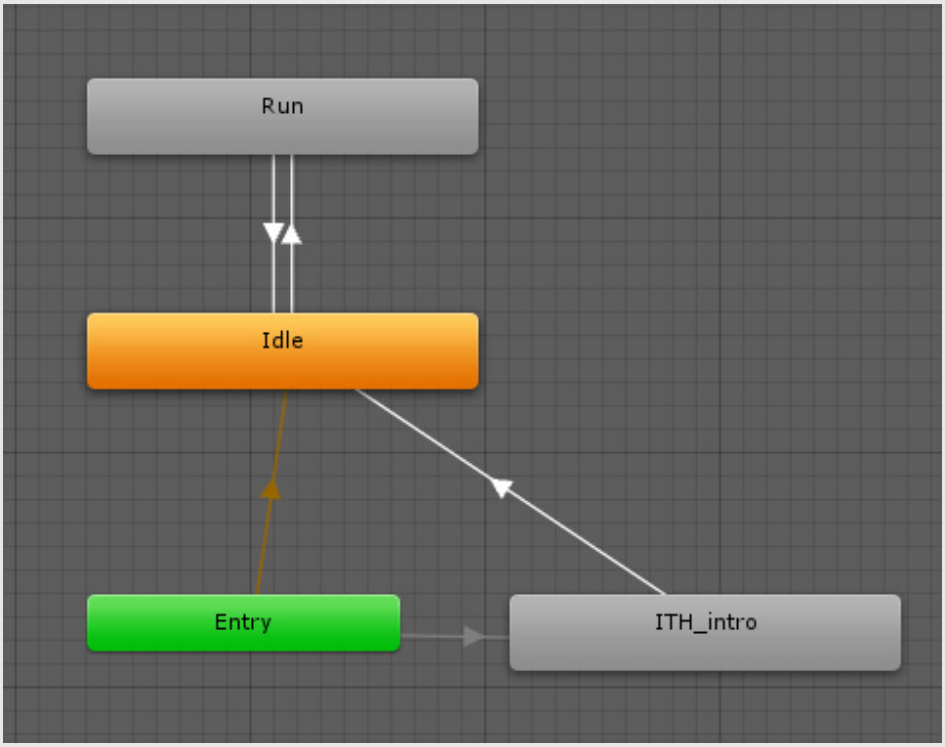
El componente **Rigidbody2D** permite dotar de físicas a un GameObject, en el podemos configurar la masa, gravedad, fijar algún eje de coordenadas, etc. Los objetos con gravedad 1 automáticamente caerán al reproducir el juego y por tanto requieren del uso de Colliders para que colisionen.

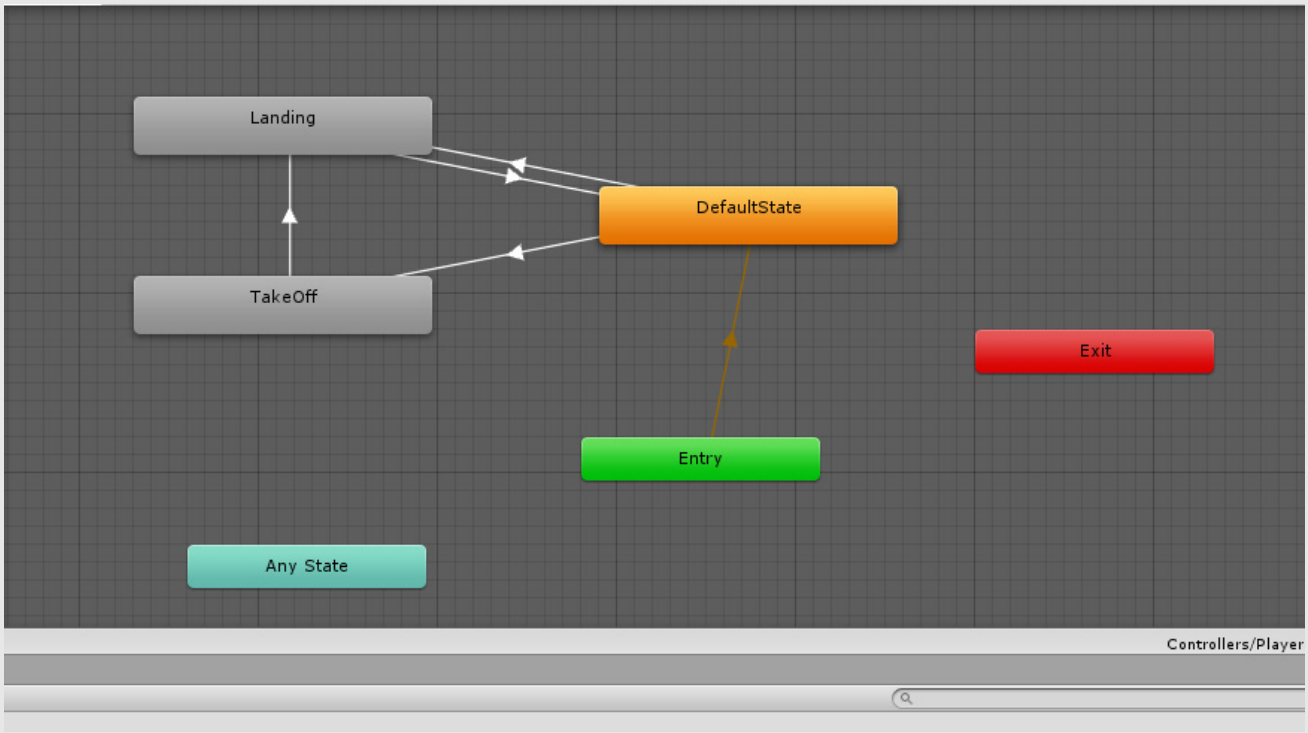
Los tipos de objetos **canvas** se utilizan para confección la UI del juego, portrait, vida del personaje, contador de estrellas y piezas, menús del juego… permiten que a diferentes resoluciones estos se escalen de forma no proporcional con el juego para permitir que siempre sean visibles.

Hay diferentes tipos de **Colliders2D** según su forma, esférica, rectangular… y pueden si marcamos "is Trigger" entonces el motor de UNITY ignorara el colider a la hora de que un objeto colisione con él ya que a través de código podemos detectar este evento y realizar los procesos que consideremos, por ejemplo cuando ITH colisiona con una estrella de mana existe un collider por script que detecta la colisión y entonces pasamos a sumar 1 al contador de estrellas. Si desmarcáramos este check el objeto pasaría a ser un objeto más sobre el apoyarnos para saltar.

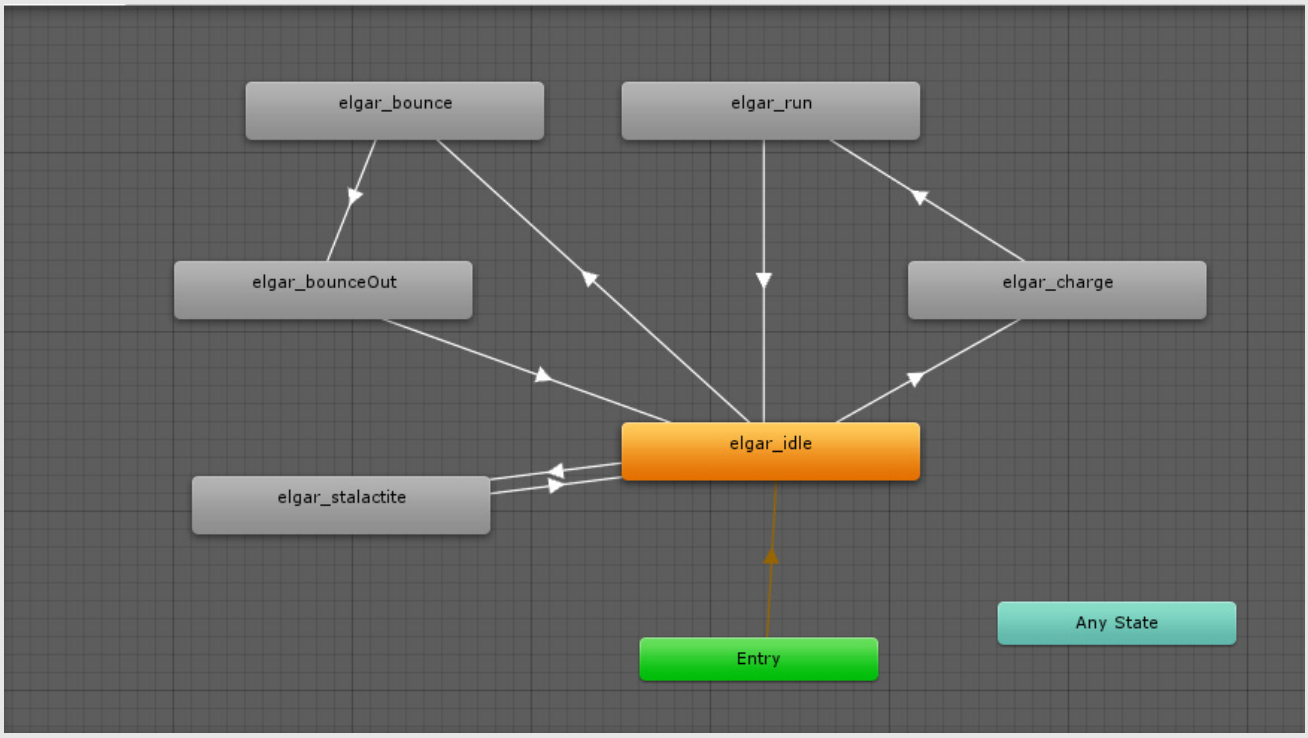
El objeto **animator** lo utilizamos para montar animaciones para el GameObject. Estas pueden ser llamadas desde script según las variables que declaremos para pasar de una animación a otra o para detectar estados de estas.

Diagramas animación personaje ITH





Diagramas animación personaje ELGAR



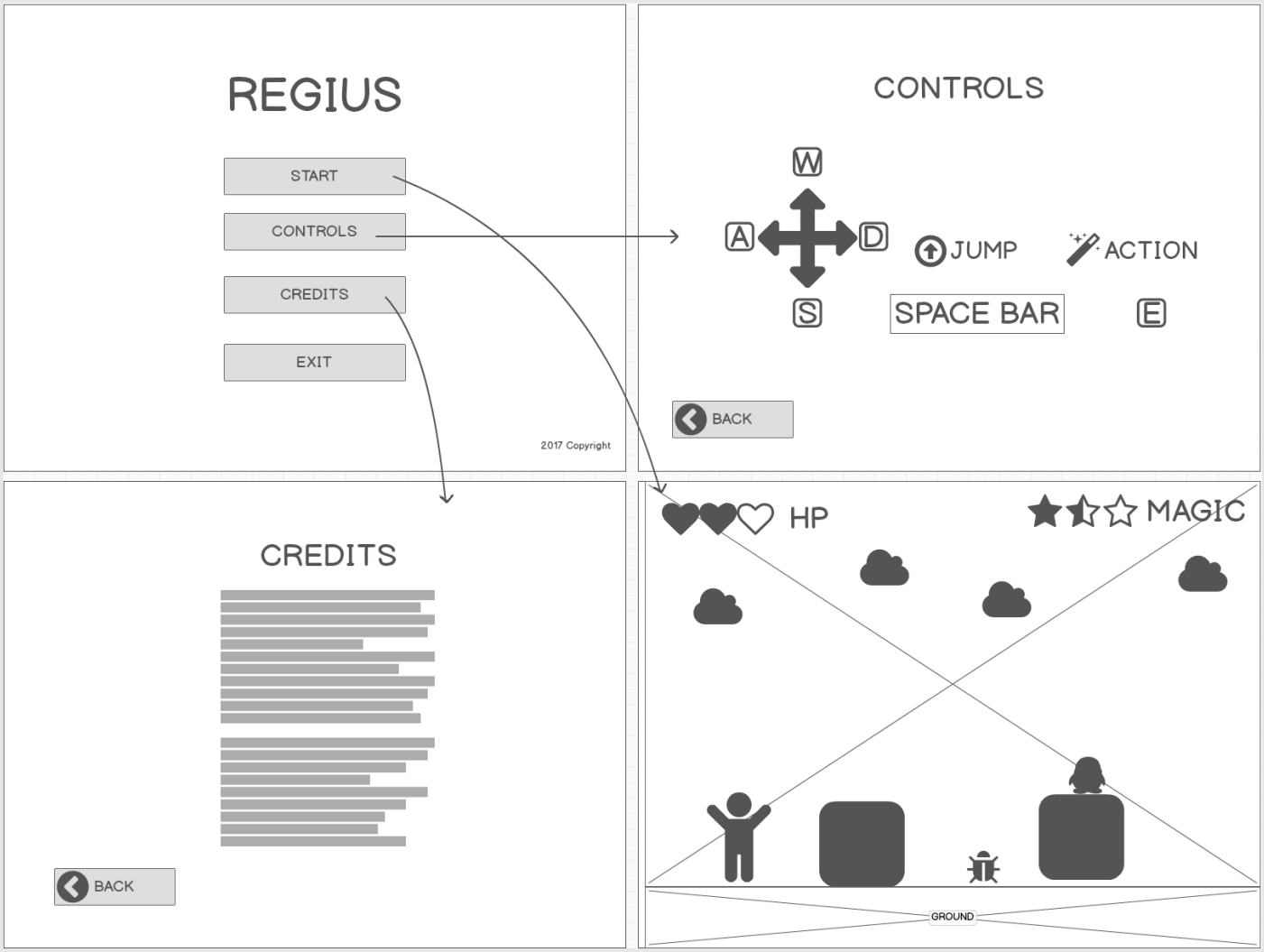
13 PROTOTIPOS

13.1 LO-FI

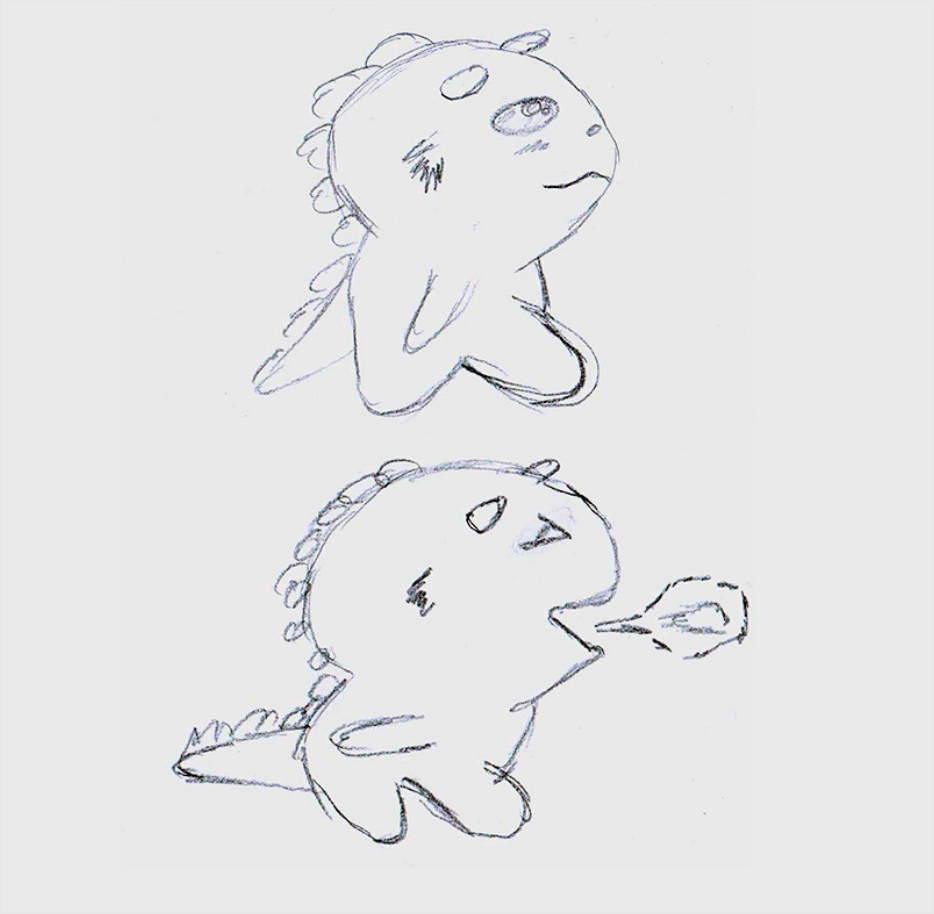
Los prototipos de baja fidelidad nos sirven para definir la **estructura de navegación** y la ubicación de los distintos elementos en la interfaz. Posteriormente se utilizará para crear los diseños finales.

La pantalla de menú inicial muestra 4 opciones:

· Start (iniciar el juego)
· Controls (muestra la configuración de controles)
· Credits (muestra los créditos del juego)
· Exit (salir)



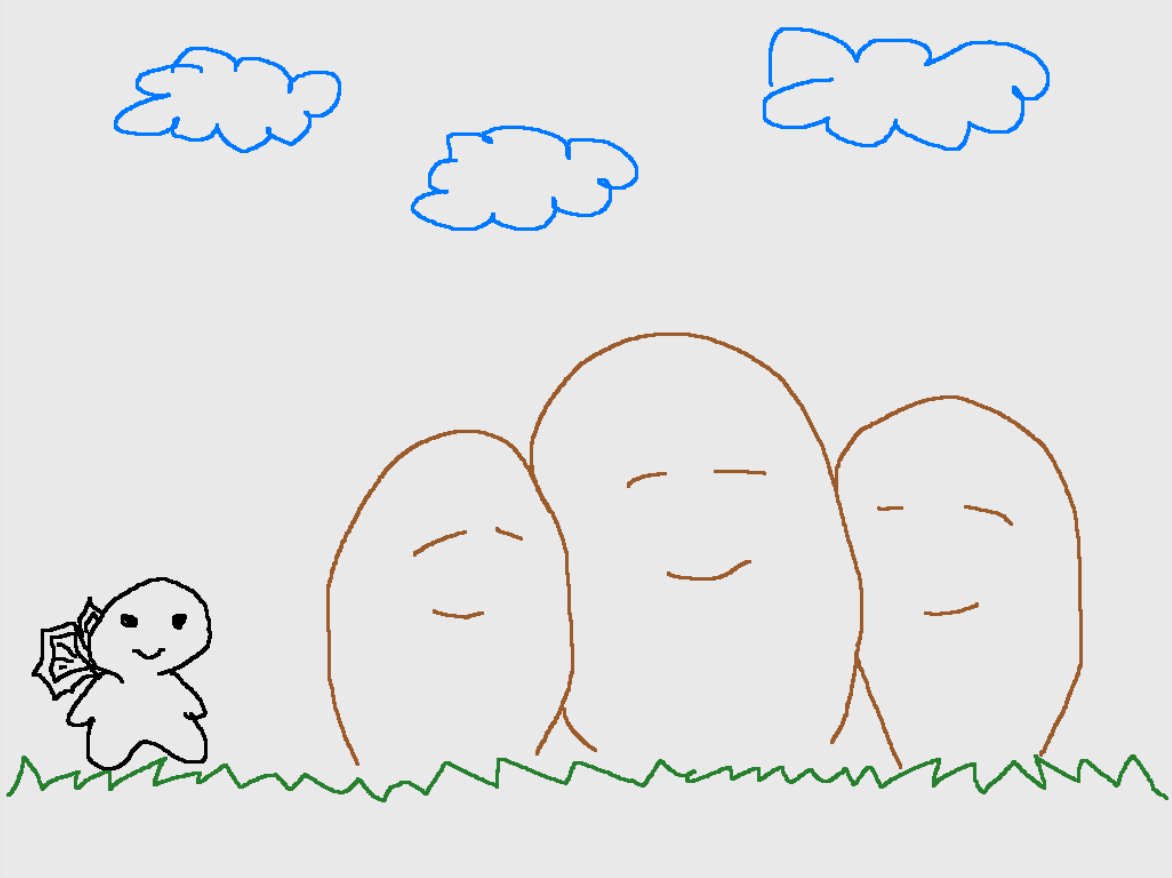
Algunos prototipos gráficos dibujados a mano alzada del personaje principal; **ITH**



Bocetos de enemigos

Resto de personajes (personalidad, expresiones)

Bocetos escenario y jefe de nivel 1 (ELGAR)

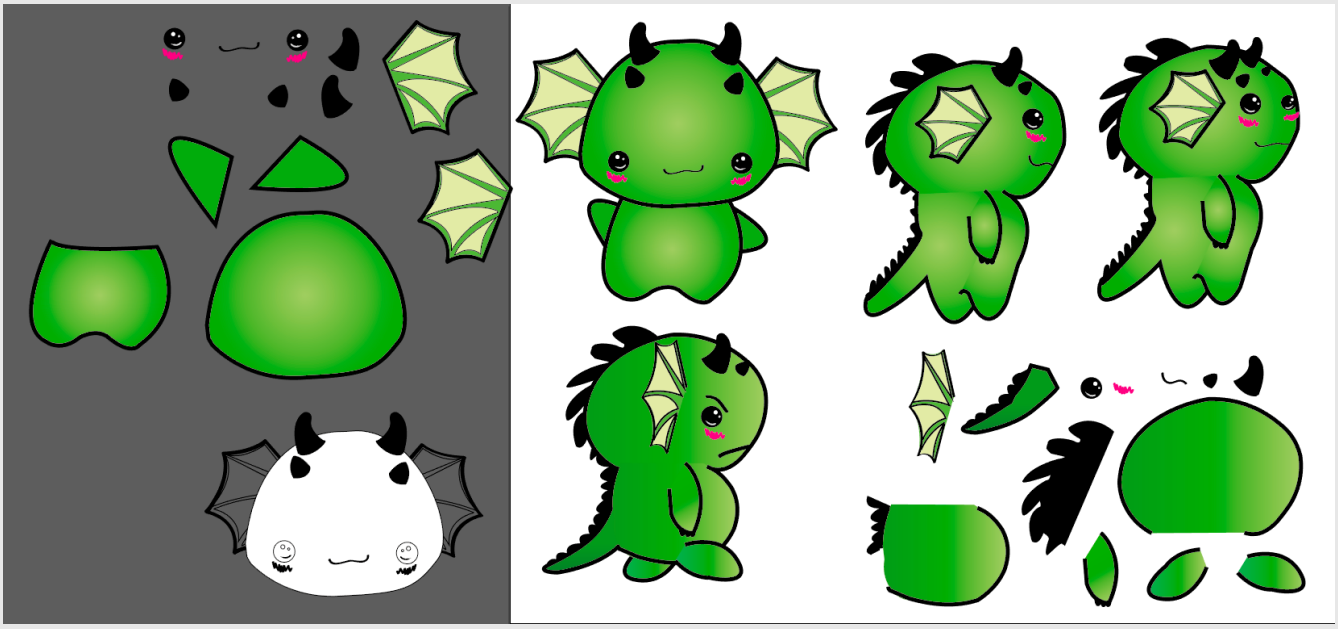


Menú principal


CONTROLS
UP W
DOWN S
LEFT A
RIGHT D
JUMP SPACE
ESC RETURN TO MAIN MENU
BACK

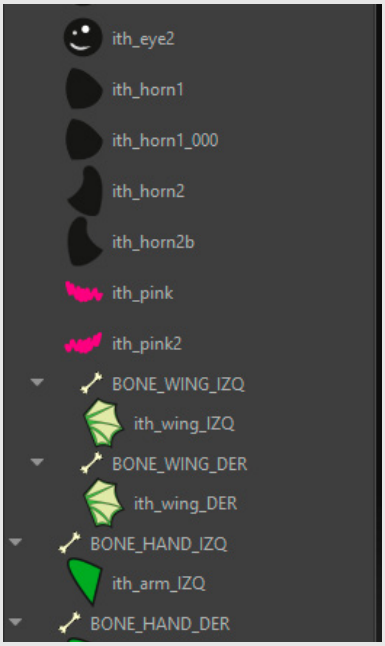
Personaje principal (ITH)

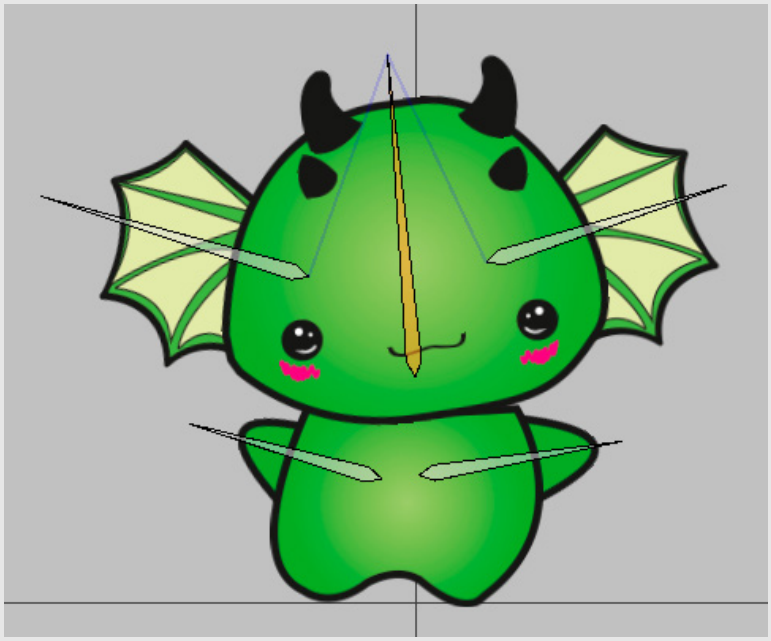
A partir del boceto realizado a mano alzada trazamos el personaje en Adobe Illustrator. En Photoshop hacemos los últimos retoques y separamos cada elemento en archivos PNG con alpha channel. Importamos los elementos en Spriter para realizar la animación de cada pieza por separado.



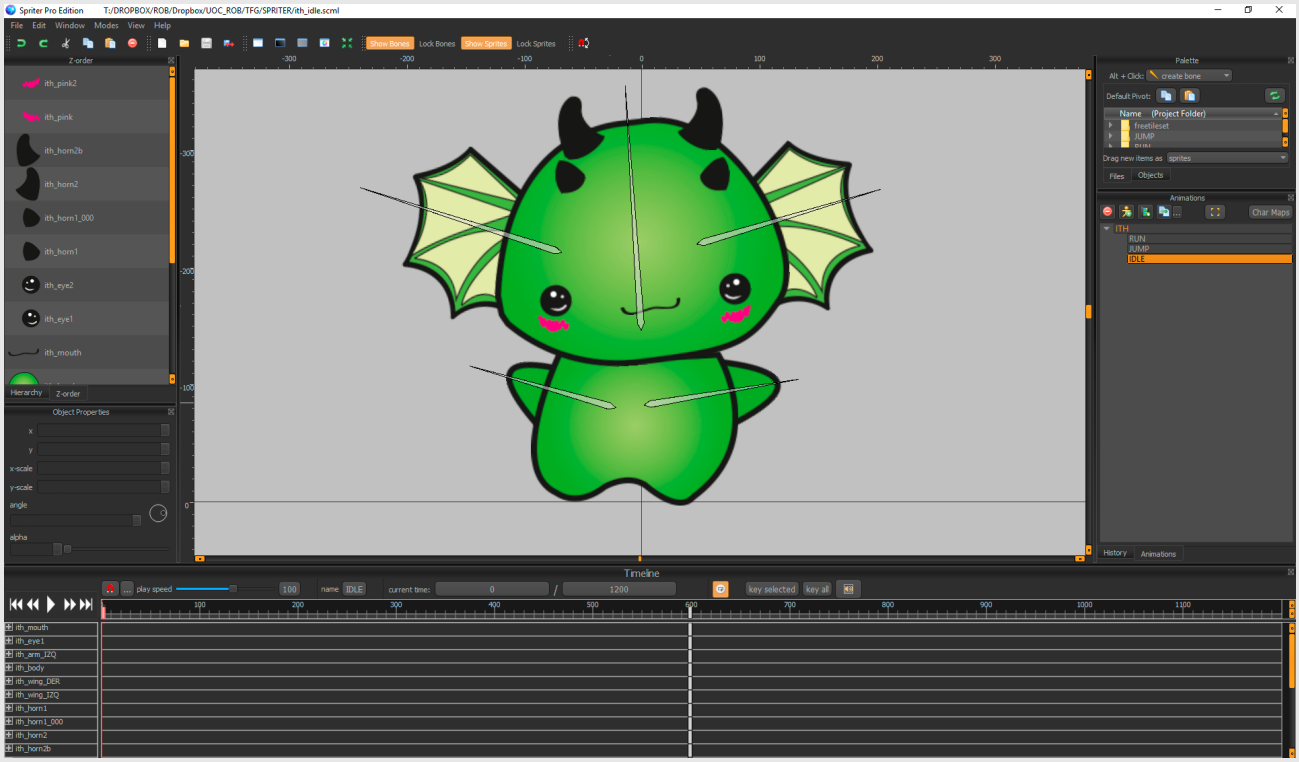
Animación personaje en Spriter

A partir del boceto realizado a mano alzada trazamos el personaje en Adobe Illustrator. En Photoshop hacemos los últimos retoques y separamos cada elemento en archivos PNG con alpha channel. Importamos los elementos en Spriter para realizar la animación de cada pieza por separado.

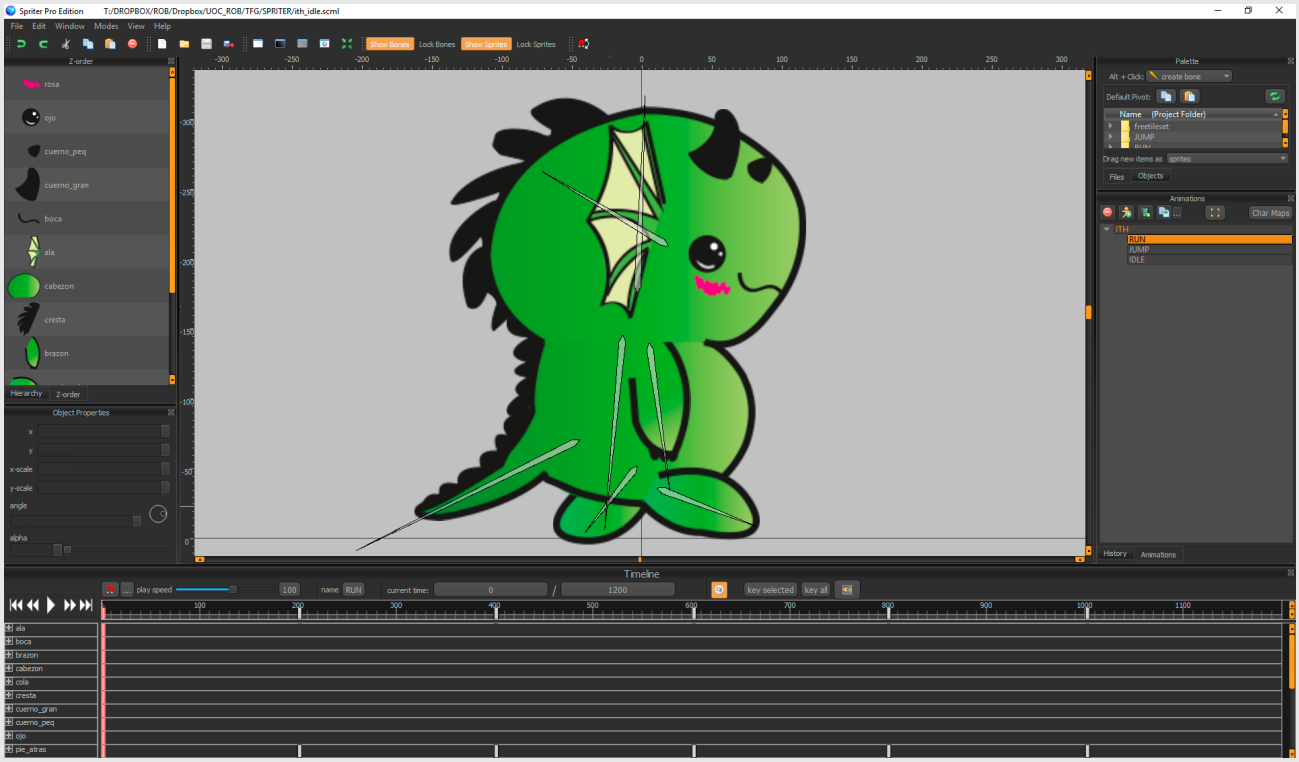




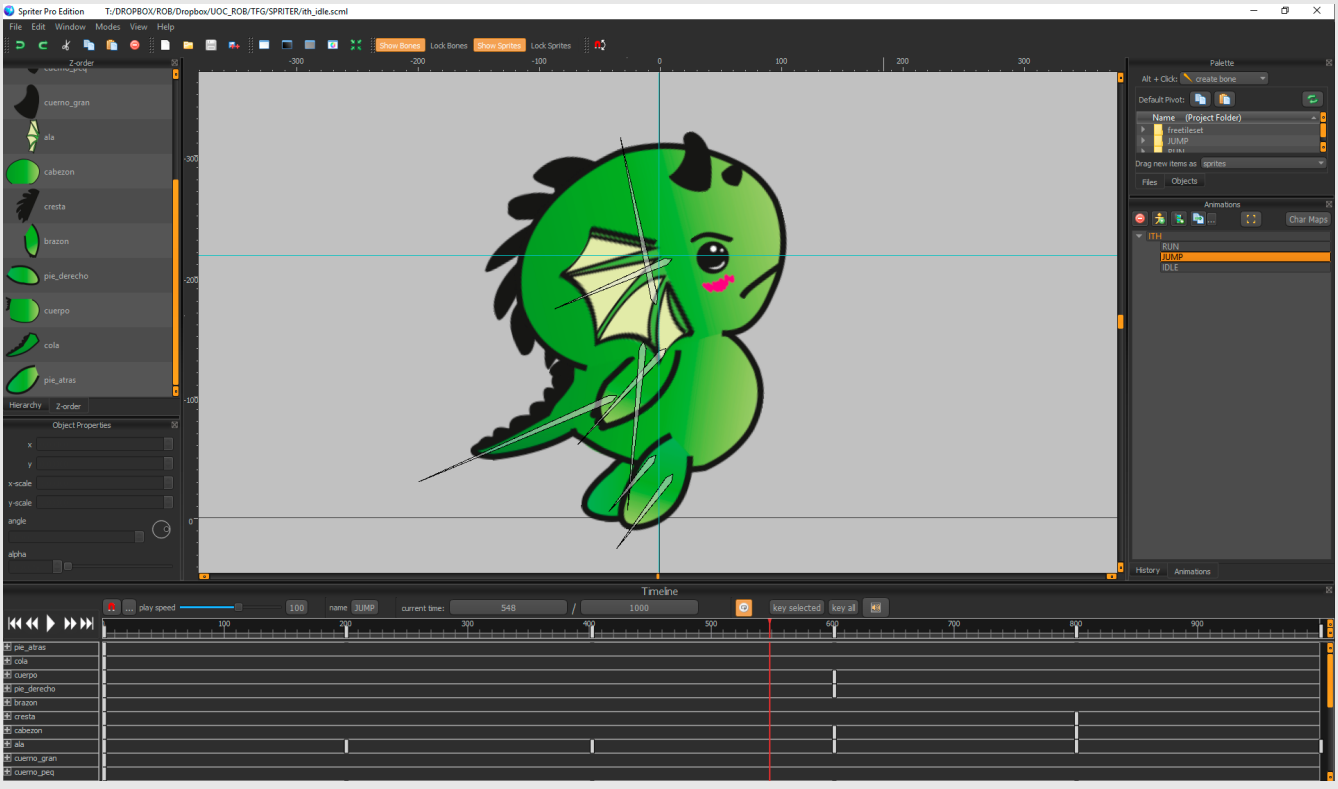
Animación IDLE



Animación RUN

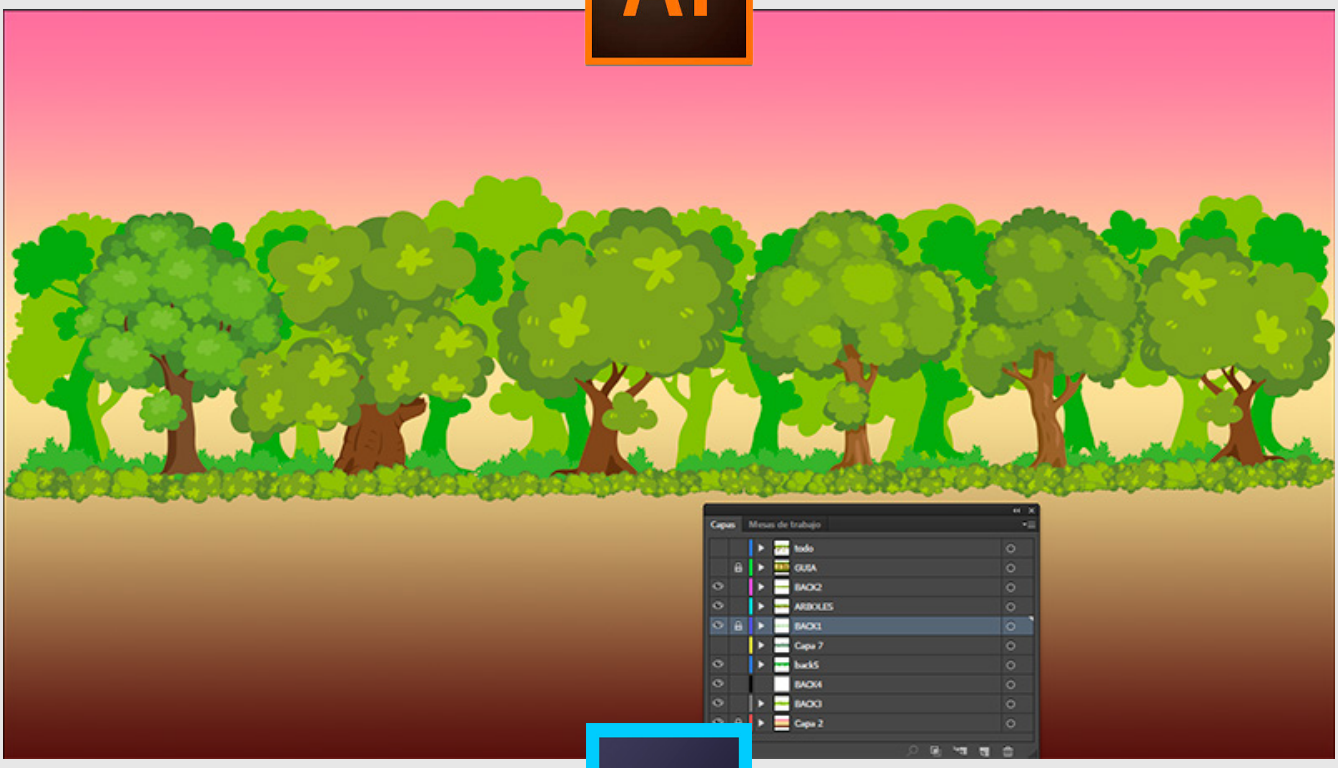


Animación JUMP



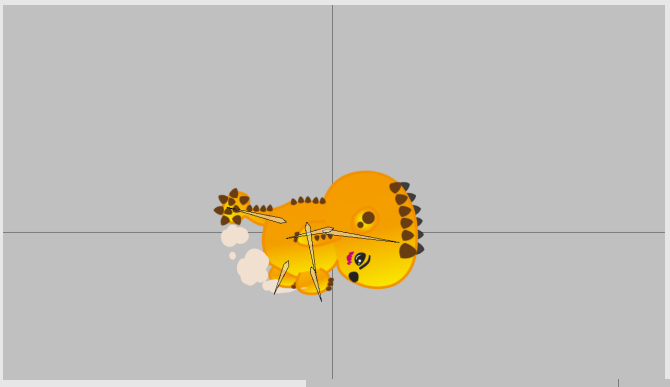
Escenario

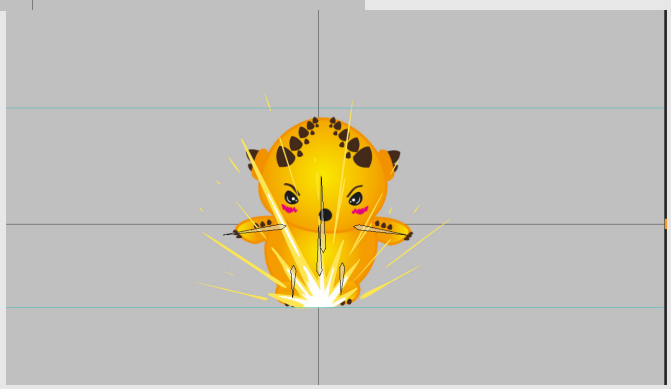
Composición en Illustrator a partir de elementos vectoriales. Edición en Photoshop y cortado en 3 bloques que se irán repitiendo a medida que el personaje avance por el escenario. De esta forma evitamos la carga de una imagen muy grande que cubra todo el ancho por donde se mueve el personaje.

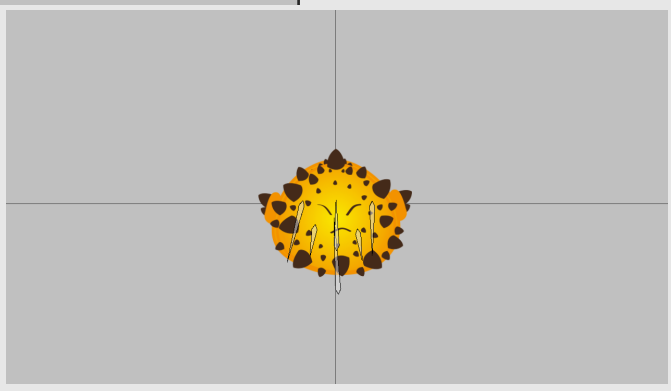


Menú principal


CREDITS
PROGRAMMING
Roberto Alcázar Alcázar
ARTS/GRAPHICS
Roberto Alcázar Alcázar
ANIMATION
Roberto Alcázar Alcázar
SPECIAL THANKS TO
Jordi Martín Romero
Víctor Alcázar Alcázar
MENU







MOVIE INTRO

Escena 1 – Conversacion inicial (5 segundos)
Se muestran 6 huevos de dragón en un nido. Hay un dialogo corte entre los dragones en el interior de cada huevo.

Escena 2 – Los huevos son robados (5 segundos)
Un extraño poder eleva el nido y transporta por el aire a los dragones que no pueden hacer nada para impedirlo.

Escena 3 – Caida de ITH (5 segundos)
Sin embargo en el transporte aéreo de los huevos 1 cae a un bosque y es perdido.

Escena 4 – 12 meses después (5 segundos)
El huevo se abre y de él sale ITH (baby dragón verde) que quiere encontrar a sus hermanos.

NOTAS

· Los huevos de dragón toman las características del medio donde se hayan incubado lo que provoca la variación de color, personalidad y habilidades de cada dragón.

· Cuando ITH se encuentra a cada hermano debe derrotarlo para liberarlo del poder que lo controla, una vez derrotado el personaje pasa a ser elegible por el jugador y podrá hacer uso de sus habilidades, necesarias para superar ciertas zonas.

· Cada huevo está escondido en diferentes tipos de terreno, con enemigos y escenarios diferentes.

Baby dragons y características:

Name	RASENTH	ELGAR	GARUGA	ITH	UK	SAPHIRA
Ability	Fireball	Charging head	Short Flight	Mobility	Lightning	Frost
Description	· Lanza proyectil de fuego que daña a enemigos y enciende fuegos	· Rompe paredes para acceder a zonas ocultas	· Al saltar planea lo que permite llegar a zonas lejanas	· Auto cura · Pasiva: Velocidad de movimiento mejorada	· Lanza relámpago que daña y aturde a enemigos · Ilumina estancias	· Congela de cerca · Pasivas: anda sobre hielo, sabe nadar
Gender	Male	Male	Female	Male	Male	Female

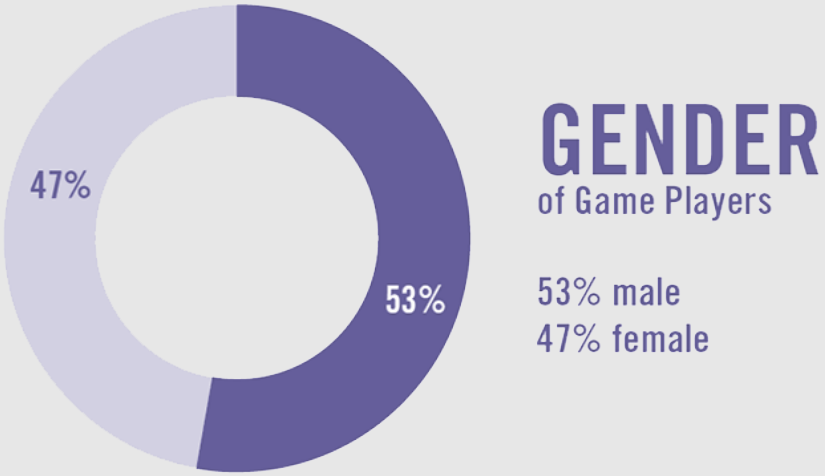
REGIUS es la palabra que conforman las iniciales de los 6 dragones. Significa “lio” en latín y también es una especie de serpiente Python, que será el monstruo final del juego.

15 PERFILES DE USUARIO

Podemos clasificar a los usuarios que juegan a videojuegos según su género, edad, formación, procedencia y frecuencia de uso o tipo de jugador.

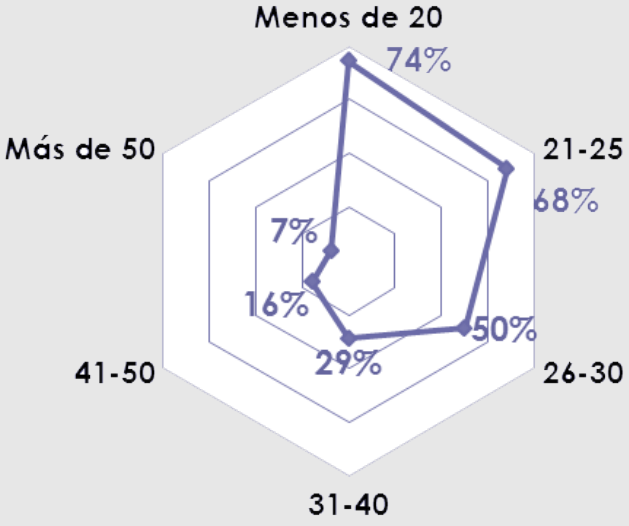
GÉNERO

Tradicionalmente los videojuegos era un medio de ocio más extendido entre el género masculino que el femenino. En la última década esta tendencia se está igualando.



EDAD

El consumidor medio de videojuegos ronda los 30, los menores de 20 son los mayores consumidores mientras que a partir de los 50 el porcentaje de usuarios es bajo.
Hay que tener en cuenta el corte generacional, ya que el consumo de videojuegos no se popularizó de forma domestica hasta los 90 y por tanto es probable que las próximas generaciones de jugadores incluyan personas de mayor edad si lo comparamos con la situación actual.

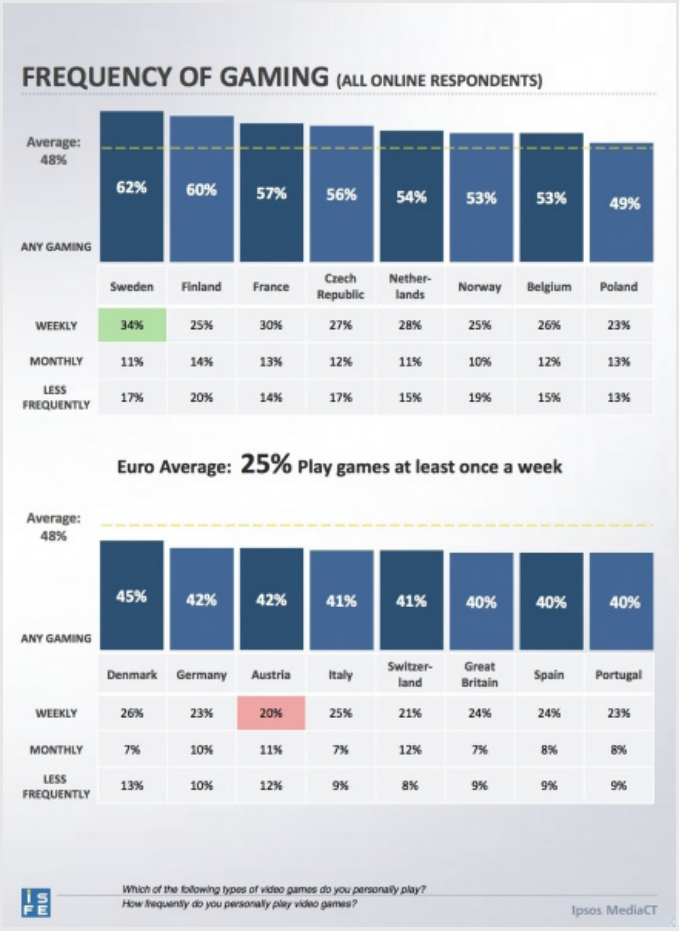


Fuente: The Competitive Intelligence Unit, 2014

TIPO DE JUGADOR

Define el tiempo que dedica al uso de videojuegos y con qué nivel de interés o esfuerzo.

- **Novel**: jugador con poca o nula experiencia. Requiere procesos de adaptación más largos.
- **Casual**: jugador que juega esporádicamente y de forma irregular. Busca mecánicas sen cillas e historias no demasiado complejas.
- **Habitual**: Dedica parte de su tiempo de ocio a los videojuegos sin buscar un grado de perfección elevado.
- **Hardcore**: Dedica gran parte de su tiempo a los videojuegos. Busca nuevos retos cons tantemente, mejorar y competir.
- **Profesional**: Los videojuegos se han convertido en su medio de vida. Buscan la excelencia y optimización máxima para mantener ese estado.



REGIUS es un juego de plataformas de sencillo manejo y gráficos con un arte infantil que se acercará a un **público joven**. Aunque está orientado a todo tipo de jugadores por sus características va más dirigido a un público **Casual/Habitual**.

16 USABILIDAD/UX

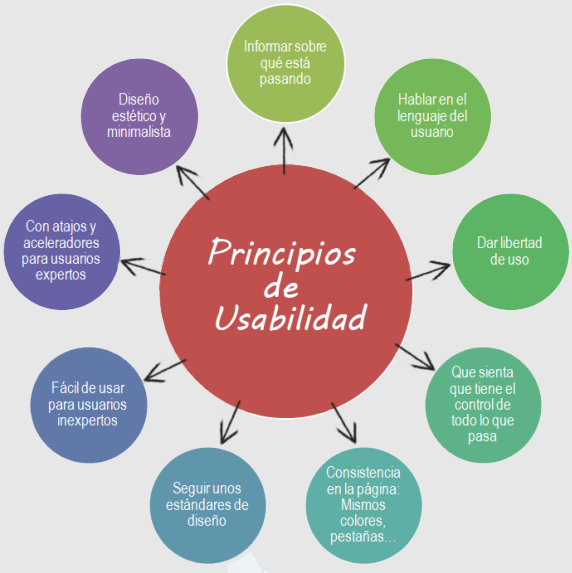
El principal objetivo de un videojuego es el **entretenimiento** y para ello el desarrollador debe asegurar que el aprendizaje de las mecánicas necesarias para su uso se aprendan de forma rápida.

Por un lado necesitamos unos **controles fáciles de entender**, y por eso debemos minimizar el número de botones a utilizar para que así sea más sencillo recordar las diferentes acciones. Una opción es utilizar los controles habituales en este tipo de juegos para asegurar una toma de control rápida con usuarios habituados a este tipo de juegos.

Por otro lado, los menús deben disponer de **elementos** visualmente **claros** y **atractivos**, la interfaz debe permitir identificar cada elemento de forma rápida para así acelerar esta fase de aprendizaje y poder dedicar el mayor tiempo al disfrute del usuario.

Por último, una vez dominada la interfaz y los controles, hay que asegurar que la **curva de dificultad** de las mecánicas dentro del juego, sea la adecuada. Es por esto que debemos evitar la sensación de frustración en el usuario en el caso de plantear situaciones excesivamente complejas sobretodo en la fase inicial de juego. Por el contrario también debemos evitar que el reto no se vuelva excesivamente sencillo ya que puede llevar a la monotonía, al aburrimiento y al abandono del juego.

El procedimiento es ir incorporando mecánicas a medida que avanza la acción de forma que el usuario tenga tiempo de asimilarlas. De la misma forma el número de enemigos y la dificultad de las zonas debe ser progresiva, incorporando puntos donde el usuario encuentre retos de mayor dificultad pero que puedan ser superados una vez entienda las mecánicas necesarias para hacerlo.



18 TESTS

El testeo de la aplicación constará de 2 fases:

FASE 1
Esta fase se realizará de forma inicial a medida que vayamos creando los sprites **gráficos** que compondrán el juego. Buscar **opiniones** de terceros nos puede servir para acabar de ajustar o modificar aquellos elementos que no concuerden o sean mejorables desde el punto de vista de otros usuarios.

Una vez definido el apartado gráfico haremos lo mismo con las **mecánicas** de movimiento y salto del personaje así como de algún enemigo. Toda esta información nos servirá para comprobar si vamos en la buena línea y para obtener puntos de vista que hayamos podido obviar.

Este testeo inicial se realizará de forma **iterativa** y contará con un muestreo de personas muy selecto, intentando buscar **perfiles profesionales** que puedan aportar ideas y opiniones de mayor valor.

FASE 2
Una vez el juego ya esté implementado se realizará una fase de testeo más profunda, buscando un muestreo mucho mayor, diverso y un tanto aleatorio. Para ello buscaremos en nuestro entorno más cercano personas de diferentes rangos de edades, sexo, interés, frecuencia de juego, etc.

Para ampliar el muestreo utilizaremos las **redes sociales** para dar a conocer el juego y obtener información de otras personas. Podemos utilizar un generador de **encuestas** online donde los usuarios entren y de forma rápida puedan contestar a una serie de preguntas que les hagamos, esto facilitaría la tarea de recopilación de información.

TESTEO VERSIÓN ALPHA

JORDI MARTÍN ROMERO
Programador Multimedia / Animador / Fotógrafo
Valoración Demo Alpha - **Muy positiva**
Observaciones a mejorar
· Las piernas del personaje podrían ser más cortas y redondas, se parecen mucho a los brazos.

VICTOR ALCÁZAR
Programador
Valoración Demo Alpha - **Positiva**
Observaciones a mejorar
· La animación de correr es demasiado lenta, “la animación patina” hay que acelerarla para que no de este efecto. (Solucionado v.A1.1)

RAUL RAMOS LÓPEZ
Programador
Valoración Demo Alpha - **Neutra**
Observaciones a mejorar
· Saltar en parado no deja desplazarte hacia los lados.
· Faltaría doble salto y que el personaje disparara.
· No hay posibilidad de reiniciar el juego cuando ocurre el bug de la colisión. (Solucionado pulsando ESC para volver al menú)

DALILA
Programadora
Valoración Demo Alpha - **Positiva**
Observaciones a mejorar
· Visualmente hay objetos que no se ven muy bien como las estrellas.
· No se aprecia dónde te puedes caer.
· Incorporar algún boucing cuando se recogen objetos.
· Cerrar el juego, aunque sea una versión demo. (Solucionado v.A1.1)

ESTHER CORRALES
Diseñadora
Valoración Demo Alpha - **Muy Positiva**
Observaciones a mejorar
· Desenfoque suave del fondo para dar efecto profundidad.
· Contrastar objetos que puedan ser recogidos.
· Modificar piernas personaje e incorporar algún efecto sutil al saltar.

TESTEO VERSIÓN BETA

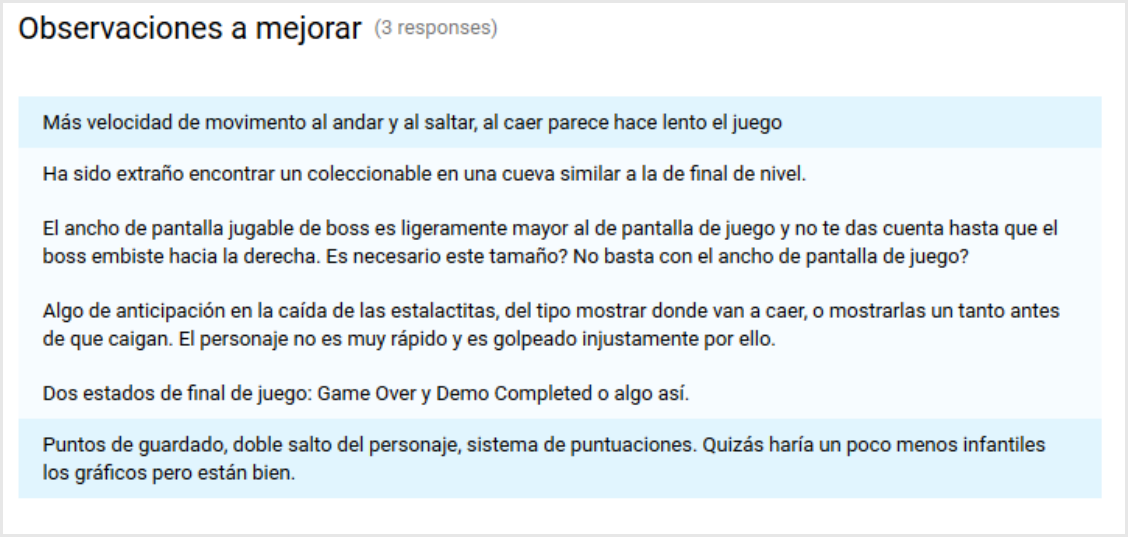
Para el testeo de la versión Beta primero la enseñe en el entorno más cercano del que obtuve un report de errores mostrados en este enlace:

https://docs.google.com/document/d/1sauiQ-i34KUqwaqX6VhjR5YkoQIZOOeguYbGC-2T7i_E/edit

Una vez corregidos la abrí a más usuarios creando una encuesta donde recoger información del tipo de usuario que ha testeado el juego y preguntando qué cosas mejorarían:

https://docs.google.com/forms/d/e/1FAIpQLScQVp3YSny6Z2Fe_WHUktaox41O1doJG-M2QRik3tda_e2If1A/viewform

Algunas observaciones registradas:



Observaciones a mejorar (3 responses)

Más velocidad de movimento al andar y al saltar, al caer parece hace lento el juego

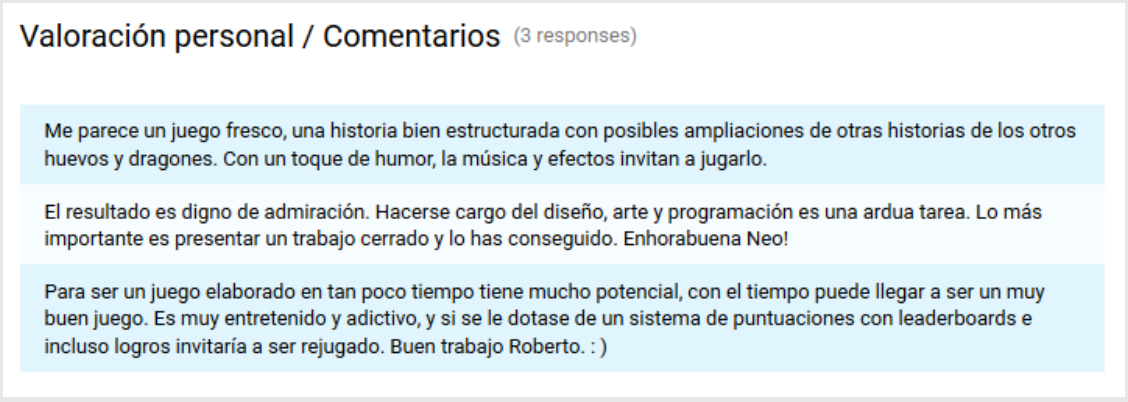
Ha sido extraño encontrar un coleccionable en una cueva similar a la de final de nivel.

El ancho de pantalla jugable de boss es ligeramente mayor al de pantalla de juego y no te das cuenta hasta que el boss embiste hacia la derecha. Es necesario este tamaño? No basta con el ancho de pantalla de juego?

Algo de anticipación en la caída de las estalactitas, del tipo mostrar donde van a caer, o mostrarlas un tanto antes de que caigan. El personaje no es muy rápido y es golpeado injustamente por ello.

Dos estados de final de juego: Game Over y Demo Completed o algo así.

Puntos de guardado, doble salto del personaje, sistema de puntuaciones. Quizás haría un poco menos infantiles los gráficos pero están bien.

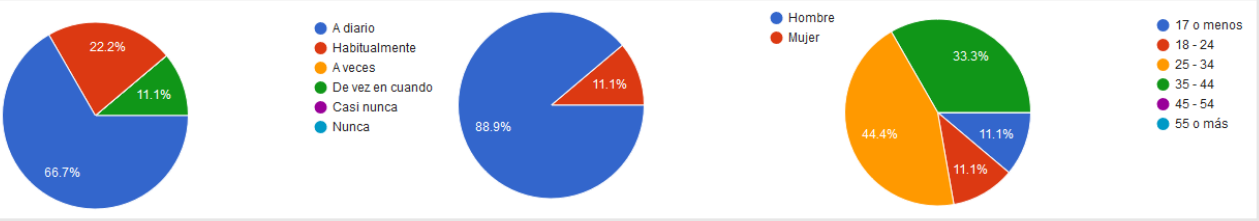


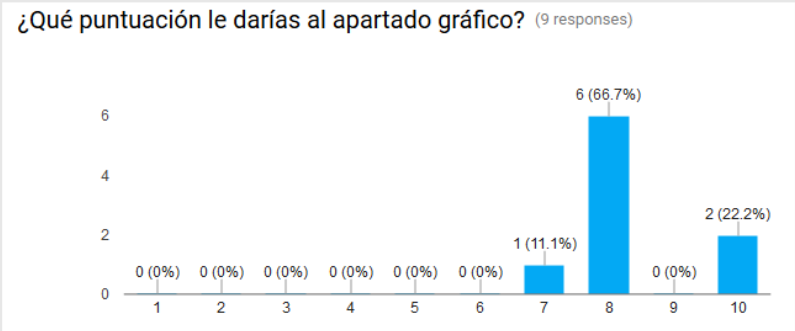
Valoración personal / Comentarios (3 responses)

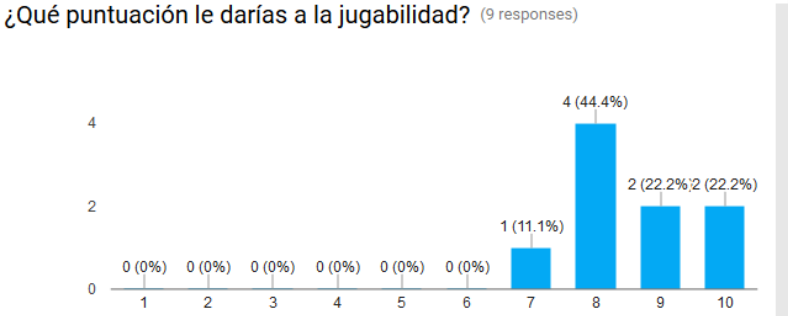
Me parece un juego fresco, una historia bien estructurada con posibles ampliaciones de otras historias de los otros huevos y dragones. Con un toque de humor, la música y efectos invitan a jugarlo.

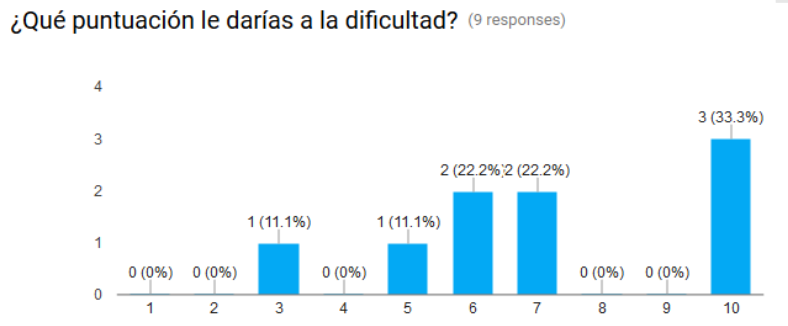
El resultado es digno de admiración. Hacerse cargo del diseño, arte y programación es una ardua tarea. Lo más importante es presentar un trabajo cerrado y lo has conseguido. Enhorabuena Neo!

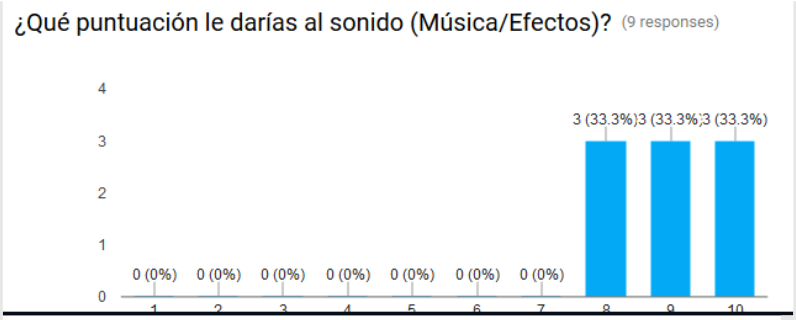
Para ser un juego elaborado en tan poco tiempo tiene mucho potencial, con el tiempo puede llegar a ser un muy buen juego. Es muy entretenido y adictivo, y si se le dotase de un sistema de puntuaciones con leaderboards e incluso logros invitaría a ser rejugado. Buen trabajo Roberto. :)

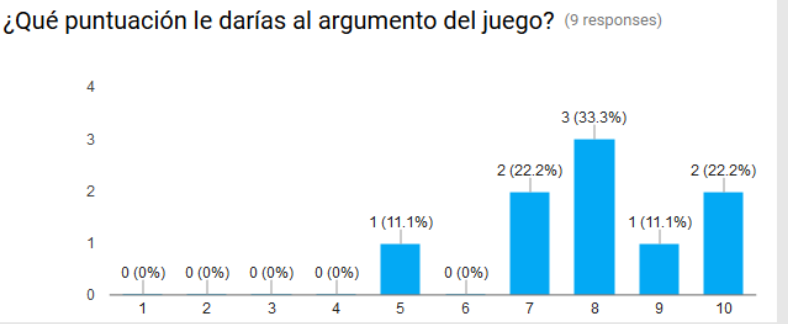

22.2%
11.1%
66.7%
A diario
Habitualmente
A veces
De vez en cuando
Casi nunca
Nunca
11.1%
88.9%
Hombre
Mujer
33.3%
44.4%
11.1%
11.1%
17 o menos
18 - 24
25 - 34
35 - 44
45 - 54
55 o más


¿Qué puntuación le darías al apartado gráfico? (9 responses)
6 (66.7%)
2 (22.2%)
1 (11.1%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
1
2
3
4
5
6
7
8
9
10


¿Qué puntuación le darías a la jugabilidad? (9 responses)
4 (44.4%)
2 (22.2%)
2 (22.2%)
1 (11.1%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
1
2
3
4
5
6
7
8
9
10


¿Qué puntuación le darías a la dificultad? (9 responses)
3 (33.3%)
2 (22.2%)
2 (22.2%)
1 (11.1%)
1 (11.1%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
1
2
3
4
5
6
7
8
9
10


¿Qué puntuación le darías al sonido (Música/Efectos)? (9 responses)
3 (33.3%)
3 (33.3%)
3 (33.3%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
1
2
3
4
5
6
7
8
9
10


¿Qué puntuación le darías al argumento del juego? (9 responses)
3 (33.3%)
2 (22.2%)
2 (22.2%)
1 (11.1%)
1 (11.1%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
1
2
3
4
5
6
7
8
9
10

19 VERSIONES DE LA APLICACIÓN

La aplicación dispondrá de las siguientes versiones:

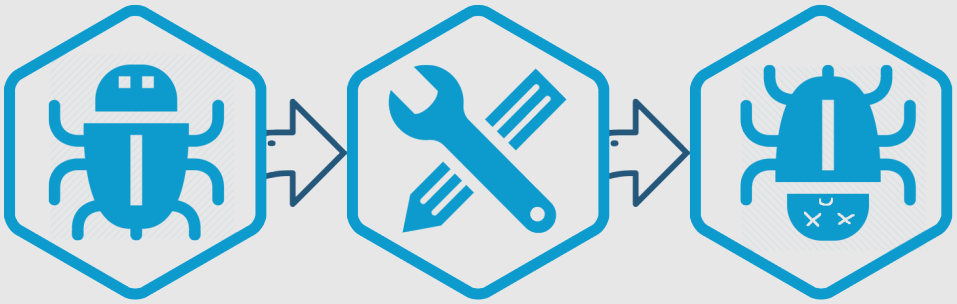
ALPHA α
Versión del juego con los primeros sprites agregados y alguna mecánica implementada del personaje principal.
(Entrega prevista 4 Diciembre 2016)

BETA β
Versión del juego completa con todas los sprites y mecánicas implementadas. Versión lista para la fase de testeo de usuarios.
(Entrega prevista 4 Enero 2017)

v. 1.0
Versión completa del juego con cambios implementados gracias al feedback de los usuarios que hayan testeado la aplicación.
(Entrega prevista 16-25 Enero 2017)

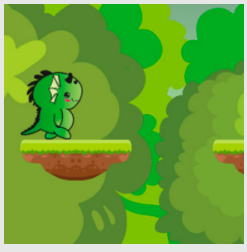
23 BUGS

En este punto analizaremos los posibles errores o comportamientos inesperados en nuestra aplicación, como afecta a su funcionamiento, la frecuencia con la que ocurren, así como sus posibles soluciones.
Estos errores suelen detectarse en la fase de testeo.



α BUGS VERSIÓN ALPHA

Colisión personaje y esquinas entorno
Descripción: A veces el personaje se queda clavado al colisionar con las esquinas de los objetos sin que el jugador pueda desplazarlo ni realizar ninguna acción.
Solución: revisar script colisiones.



Plataforma móvil
Descripción: al subirse el personaje no se mueve con ella.
Solución: sumar la velocidad de la plataforma a la del personaje.

β BUGS VERSIÓN ALPHA

Queda pendiente la exportación del proyecto a WebGL que debido a un bug en el código no es posible.

25 PRESUPUESTO

Podemos desglosar los costes de este proyecto en 3 areas:

Hardware

Se ha utilizado un **PC de sobremesa**, **2 monitores** y una **tableta Wacom** para el dibujo de los diseños. Ya disponía de todo el material necesario y por tanto no ha supuesto un coste adicional. En el caso de tener que adquirirlo habría supuesto entorno a los 1500€.

Software

Dado que disponemos de licencias para la suite de **Adobe** y para la versión gratuita de **Unity** tampoco ha supuesto un coste el software. En el caso de que el juego se comercializase necesitaríamos adquirir una licencia comercial de Unity y otra licencia de **Photoshop** e **Illustrator**.

Recursos humanos

Este proyecto ha sido desarrollado en su totalidad por mi y por tanto no ha supuesto tampoco un coste económico. En un escenario de desarrollo del juego por un equipo multidisciplinar podríamos desglosar los siguientes costes por perfil profesional y número de horas:

Diseñador \ Artista	30€ / hora	100 horas	3000 €
Programador	40€ / hora	150 horas	6000 €
Animador	35€ / hora	100 horas	3500 €
Diseñador de niveles	20€ / hora	50 horas	1000 €
Técnico de sonido	20€ / hora	15 horas	300 €
Especialista en marketing	15€ / hora	25 horas	375 €
			Total: 14750 €

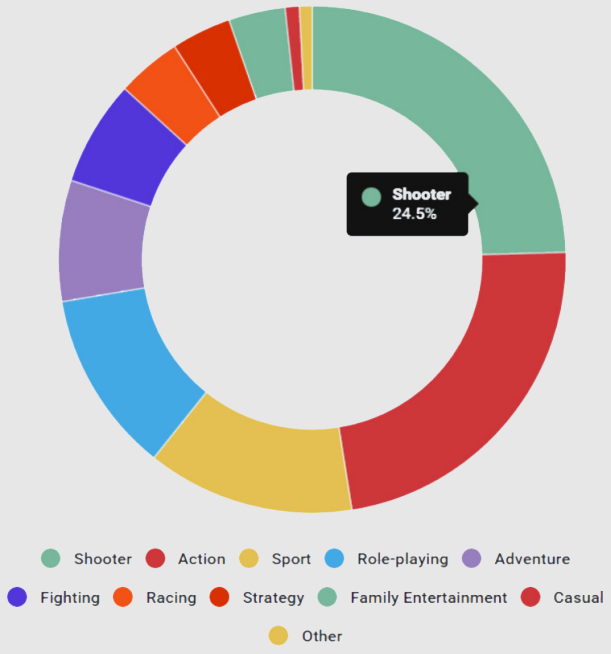
26 ANÁLISIS DE MERCADO

Aunque la demanda de títulos en el mercado de videojuegos va en aumento, existe una amplísima oferta, por lo que es muy difícil hacerse un hueco. Son las **grandes productoras** las que se reparten la mayor parte del pastel, ya que ponen en el mercado proyectos **AAA** a los que han destinado muchos recursos para su desarrollo y costosas campañas de marketing.

Por este motivo para un desarrollador *indie* o un pequeño estudio es prácticamente imposible alcanzar estos objetivos, por lo que hay que buscar posicionarse en el porcentaje de usuarios que busca otro tipo de entretenimiento.

En cuanto a los medios de distribución, por un lado la proliferación de **dispositivos inteligentes** como **smartphones y tabletas** ha ampliado el espectro de hardware donde correr videojuegos, tradicionalmente reproducidos en ordenadores y consolas. Por otro lado también han proliferado los sitios web que sirven de distribuidores, ya sea vía plataformas como **Steam** o redes sociales como **Facebook** con la incursión de juegos en flash y html5.

A pesar de que competir en el sector del juego indie está complicado por el alto volumen de oferta y no tan elevada demanda, el sector tiene su público, así que de conseguir un producto original y de suficiente calidad se puede sobresalir y ser económicamente viable. Casos como **Limbo, Super Meat boy, Braid o Minecraft** son grandes ejemplos de que con escasos recursos se pueden crear grandes obras.



http://www.gamingobserver.com/20152016-us-video-and-computer-game-industry-overview-report/

27 MÁRQUETIN Y VENTAS

Para que el proyecto sea viable económicamente necesitamos darlo a conocer. Hoy en día existen miles de juegos en el mercado con características similares por lo que es muy complicado dar a conocer el juego sin una campaña de marketing adecuada.

Existen plataformas como **STEAM** que son utilizadas por millones de usuarios y donde podemos dar a conocer nuestro juego. Para conseguirlo se creó **Steam Greenlight** que busca apoyar a los desarrolladores de juegos independientes. Gracias a esta iniciativa se dan a conocer una oferta de juegos y es la comunidad la que decide, a través de votaciones, que juegos pasan el filtro y cuáles no.

Para que nuestro juego figure dentro de Steam Greenlight necesitaremos preparar una portada, 4 o más capturas, un video del juego en funcionamiento y al menos 2 parrafos que describan el juego y los requisitos de sistema. Los usuarios de Steam podrán entonces acceder a la página propia del juego y votar Si o No según lo interesados que estén en el juego. Una vez alcanzados un número suficiente de votos positivos Valve se pone en contacto con los desarrolladores para definir un calendario de entrega (si el juego no está acabado) y un precio para el producto. Este proceso tiene un coste de $100.

Aunque la iniciativa es buena a la práctica las votaciones de Greenlight reflejan más las habilidades publicistas que no la calidad del juego en sí. Es por tanto necesario dar a conocer el juego en todos los medios de los que dispongamos, como **foros** y **redes sociales** (F**acebook, Twiter, Google+**...) que apunten a nuestra página de Steam Greenlight.

Encontramos otras vías de comercialización en las plataformas que **Xbox Arcade**, Sony o Nintendo ofrecen para desarrolladores independientes. Previamente debemos evaluar los costes que suponen los ports a estas plataformas, condiciones y licencias necesarias.

En el caso de que el producto final sea de calidad y estemos convencidos de su éxito podemos dedicar mayores recursos económicos en darlo a conocer a través de **banners** publicitarios, reviews de **youtubers** dedicados y otros medios.

A1 ENTREGABLES DEL PROYECTO

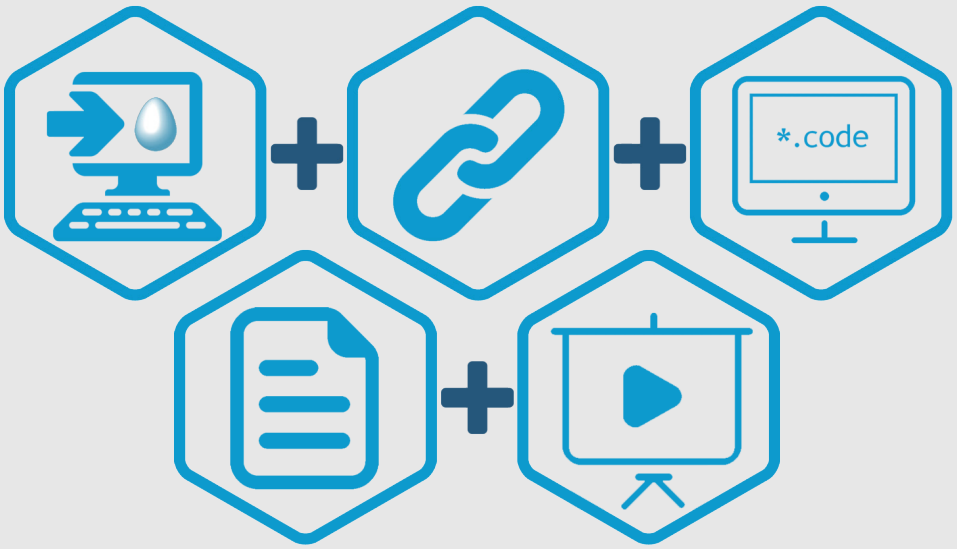
Se entregaran los siguientes ítems:

En relación con el videojuego REGIUS

· Setup.exe -> Instalador para PC del juego
· Link web -> Link a la versión para browser del juego
· Proyecto Unity -> Codigo fuente del proyecto y resto de recursos gráficos y sonoros.

Resto de documentos

· Memoria del proyecto
· Video presentación del proyecto



AnimateObject.cs

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

public class AnimateObject : MonoBehaviour {
	// Update is called once per frame
	private Vector3 starpos;
	void Start(){
		starpos = GameObject.FindGameObjectWithTag (“IthStars”).transform.position;
	}
	void Update () {
		if (gameObject.GetComponent<BoxCollider2D> ().enabled == false) {
			//gameObject.GetComponent<AudioSource> ().Play ();
			starpos = GameObject.FindGameObjectWithTag (“IthStars”).transform.position;
			gameObject.transform.position = Vector3.Lerp (gameObject.transform.position, starpos, Time.
deltaTime*5);
			StartCoroutine(destroyStar(0.55f));
		}

	}
	IEnumerator destroyStar(float howmanyseconds){
		yield return new WaitForSeconds (howmanyseconds);
		GameObject.FindWithTag(“ManaItems”).GetComponent<AudioSource>().Play();
		Destroy (gameObject);
	}
}
```

AnimatePuzzle.cs

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

public class AnimatePuzzle : MonoBehaviour {
	// Update is called once per frame
	private Vector3 starpos;
	void Start(){
		starpos = GameObject.FindGameObjectWithTag (“UIPuzzle”).transform.position;
	}
	void Update () {
		if (gameObject.GetComponent<BoxCollider2D> ().enabled == false) {
			//gameObject.GetComponent<AudioSource> ().Play ();
			starpos = GameObject.FindGameObjectWithTag (“UIPuzzle”).transform.position;
			gameObject.transform.position = Vector3.Lerp (gameObject.transform.position, starpos, Time.
deltaTime*5);
			StartCoroutine(destroyStar(0.55f));
		}

	}
	IEnumerator destroyStar(float howmanyseconds){
		yield return new WaitForSeconds (howmanyseconds);
		//GameObject.FindWithTag(“ManaItems”).GetComponent<AudioSource>().Play();
		Destroy (gameObject);
	}
}
```

BarScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class BarScript : MonoBehaviour {
	[SerializeField]
	private float fillAmount;
	// Use this for initialization
	[SerializeField]
	private Image content;

	void Start ()
	{
		content.fillAmount = fillAmount;
	}
}
```

bossWalls.cs

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

public class bossWalls : MonoBehaviour {
	void OnTriggerEnter2D(Collider2D col)
	{
		if (col.tag == "Elgar") {
			gameObject.GetComponent<AudioSource> ().Play ();
		}
	}
}
```

CameraEffects.cs

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

public class CameraEffects : MonoBehaviour {
   private float elapsedTime, linearForce, rotationForce;
   private bool isShaking;

	// Use this for initialization
	void Start () {
      isShaking = false;

   }

	// Update is called once per frame
	void Update () {
		if(isShaking)
      {
         ShakeCamera();
      }
	}

   public void Shake(float time, float linearForceMagnitude, float rotationForceMagnitude)
   {
      if(!this.isShaking)
      {
         this.isShaking = true;
         this.linearForce = linearForceMagnitude;
         this.rotationForce = rotationForceMagnitude;
         StartCoroutine(RemoveShake(time));
      }
   }
```

```
private IEnumerator RemoveShake(float t)
   {
      //Wait for camera shake
      yield return new WaitForSeconds(t);

      //Remove it slowly
      float time = 0.0f;
      float maxTime = t / 2;
      float initialLinearForce = linearForce;
      float initialRotationForce = rotationForce;
      while (time < 1)
      {
         time += Time.deltaTime / maxTime;
         linearForce = Mathf.Lerp(initialLinearForce, 0, time);
         rotationForce = Mathf.Lerp(initialLinearForce, 0, time);
         //Debug.Log("LinearForce: " + linearForce);
         yield return null;
      }
      this.isShaking = false;
   }

   private void ShakeCamera()
   {
      Vector2 shake = UnityEngine.Random.insideUnitCircle * linearForce;
      Vector3 newPosition = transform.localPosition;
      newPosition.x = shake.x;
      newPosition.y = shake.y;
      transform.localPosition = newPosition;

      float shakeRotation = UnityEngine.Random.Range(-linearForce, rotationForce);
      transform.localRotation = Quaternion.Euler(0f, 0f, shakeRotation);
   }
}
```

CameraFollow.cs

```
using UnityEngine;
using System.Collections;

public class CameraFollow : MonoBehaviour {
        [SerializeField]
        private float xMax;
        [SerializeField]
        private float yMax;
        [SerializeField]
        private float xMin;
        [SerializeField]
        public float yMin;

        [SerializeField]
        public GameObject targetToFollow;

        public Transform target;
        // Use this for initialization
        void Start () {
                //target = GameObject.FindGameObjectWithTag("Player").transform;
                target = targetToFollow.transform;
        }

        // Update is called once per frame
        void LateUpdate()
        {
                transform.position = new Vector3 (Mathf.Clamp (target.position.x, xMin, xMax), Mathf.Clamp (target.
position.y, yMin, yMax), transform.position.z);
        }
}
```

CameraSub.cs

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

public class CameraSub : MonoBehaviour {
	[SerializeField]
	private float posy;
	//public GameObject findcam;

	void OnTriggerEnter2D(Collider2D col){
		GameObject.FindWithTag (“MainCamera”).GetComponent<CameraFollow> ().yMin = posy;
	}
}
```

CharacterElgar.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
using System;

public class CharacterElgar : MonoBehaviour {
	private enum ElgarState { CHARGE, STALACTITE, BOUNCE };
   private ElgarState elgarState;

   [Header(“State and movement”)]

   public int maxCharges = 1;
   public float maxBouncingTime = 5.0f;
   public int maxStalactites = 1;


   [Header(“Physic materials”)]
   public PhysicsMaterial2D bouncingMaterial;
   public PhysicsMaterial2D stopBouncingMaterial;

   private Rigidbody2D myRigidbody;
	private Animator myAnimator;
   private float speed = 0f;
   private float direction = -0.1f;
   private float defaultGravityScale = 2.25f;
   private Transform startPoint;
   private bool isTransitioningState;

   #region FSM_VARIABLES
   private int chargesCount, stalactitesCount;
   private bool hasBounced, hasStalactited, hasChangedDirection;
   private float bouncingTime;
   #endregion

	private GameObject objEscalactita1, objEscalactita2, objEscalactita3, objEscalactita4, objEscalactita5;
   private Transform stopZone1, stopZone2;
   private GameObject frontColliderTrigger, sphereColliderTrigger, vulnerableTrigger;
	private GameObject soundShout;
   //
  [SerializeField]
   public bool bossBegin;
   private GameObject player;


  void Start ()
   {
      // STOP PLAYER CONTROL ON START
      player = GameObject.FindGameObjectWithTag(“Player”);
```

```
    soundShout = GameObject.FindGameObjectWithTag("SoundElgarShout");
                    objEscalactita1 = GameObject.FindGameObjectWithTag ("Escalactita1");
                    objEscalactita2 = GameObject.FindGameObjectWithTag ("Escalactita2");
                    objEscalactita3 = GameObject.FindGameObjectWithTag ("Escalactita3");
                    objEscalactita4 = GameObject.FindGameObjectWithTag ("Escalactita4");
                    objEscalactita5 = GameObject.FindGameObjectWithTag ("Escalactita5");
    stopZone1 = GameObject.FindGameObjectWithTag("StopZone1").GetComponent<Transform>();
    stopZone2 = GameObject.FindGameObjectWithTag("StopZone2").GetComponent<Transform>();

    vulnerableTrigger = transform.FindChild("VulnerableTrigger").gameObject;
    frontColliderTrigger = transform.FindChild("FrontCollider").gameObject;
    sphereColliderTrigger = transform.FindChild("BounceCollider").gameObject;

    startPoint = GameObject.Find("ElgarColliders/InitialPoint").GetComponent<Transform>();
    isTransitioningState = false;
    myRigidbody = GetComponent<Rigidbody2D> ();
                    myAnimator = GetComponent<Animator> ();
    myAnimator.SetBool("Stoped", true);
    InitialiseElgarToState(ElgarState.CHARGE);

}

private void InitialiseElgarToState(ElgarState state)
{
    elgarState = state;
    hasChangedDirection = false;
    isTransitioningState = false;
    hasBounced = false;
    hasStalactited = false;
    bouncingTime = 0.0f;
    stalactitesCount = 0;
    chargesCount = 0;
    transform.rotation = Quaternion.Euler(Vector3.zero);
    transform.GetComponent<BoxCollider2D>().enabled = true;
    transform.GetComponent<CircleCollider2D>().enabled = false;
    frontColliderTrigger.SetActive(false);
    sphereColliderTrigger.SetActive(false);
    vulnerableTrigger.SetActive(true);
}

void Update()
{
    if (bossBegin) {
        UpdateElgarStateMachine();
    }
}

private void UpdateElgarStateMachine()
{
    // Charge 'maxCharges' times - Bounce for 'maxBouncingTime' seconds - Stalactite during 'maxStalactiteTime'
    if(!isTransitioningState)
    {
        switch (elgarState)
        {
                                case ElgarState.CHARGE:
            if (chargesCount >= maxCharges)
            {
                isTransitioningState = true;
                Debug.Log("[Elgar] Changing state to Bounce");
                StartCoroutine(SetNextState(ElgarState.BOUNCE));
            }
            break;
          case ElgarState.BOUNCE:
            if ((Time.time - bouncingTime) >= maxBouncingTime)
            {
                isTransitioningState = true;
                Debug.Log("[Elgar] Changing state to Stalactite");
                StartCoroutine(SetNextState(ElgarState.STALACTITE));
            }
            break;
```

```
            case ElgarState.STALACTITE:
                if (stalactitesCount >= maxStalactites)
                {
                    isTransitioningState = true;
                    Debug.Log("[Elgar] Changing state to Charge");
                    StartCoroutine(SetNextState(ElgarState.CHARGE));
                }
                break;
        }
    }
}

void FixedUpdate () {
    if (bossBegin)
    {
        if (!isTransitioningState)
        {
            switch (elgarState)
            {
                case ElgarState.CHARGE:
                    UpdateDirection();
                    myAnimator.SetBool("Stoped", false);
                    Charge();
                    break;
                case ElgarState.STALACTITE:
                    Stalactite();
                    break;
                case ElgarState.BOUNCE:
                    Bounce();
                    break;
            }
        }
    }
        }

private void UpdateDirection()
{
    float distanceToCollider1 = Vector2.Distance(transform.position, stopZone1.position);
    float distanceToCollider2 = Vector2.Distance(transform.position, stopZone2.position);

    if (distanceToCollider1 >= distanceToCollider2) ChangeDirection(-0.1f);
    else                                            ChangeDirection(0.1f);
}

void OnTriggerEnter2D(Collider2D col)
{
    CheckCollider(col);
}

private void Stalactite()
{
    if(!hasStalactited)
    {
        Debug.Log("Stalactite");
        ScalactiteFall();
        hasStalactited = true;
    }
}

private void Bounce()
{
    if (!hasBounced)
    {
        vulnerableTrigger.SetActive(false);
        sphereColliderTrigger.SetActive(true);
        myAnimator.SetBool("Bouncing", true);
        bouncingTime = Time.time;
        GetComponent<BoxCollider2D>().enabled = false;
```

```
	[SerializeField]
	private Transform transformB;
	// Use this for initialization
	void Start () {
		posA = childTransform.localPosition;
		posB = transformB.localPosition;
		nexPos = posB;
	}

	// Update is called once per frame
	void Update () {
		Move ();
	}
	private void Move()
	{
		childTransform.localPosition = Vector3.MoveTowards(childTransform.localPosition, nexPos, speed * Time.
deltaTime);
		if (Vector3.Distance (childTransform.localPosition, nexPos) <= 0.1)
		{
			ChangeDestination ();
		}
	}
	private void ChangeDestination()
	{
		nexPos = nexPos != posA ? posA : posB;
	}
}

GetComponent<CircleCollider2D>().sharedMaterial = bouncingMaterial;
        GetComponent<CircleCollider2D>().enabled = true;
        Invoke("AddBounceForce", 0.5f);
        hasBounced = true;
    }
  }

  private void AddBounceForce()
  {
    myRigidbody.AddForce(new Vector2(25, 60), ForceMode2D.Impulse);
  }

  private void Charge()
  {
    if (!frontColliderTrigger.activeSelf)
    {
      frontColliderTrigger.SetActive(true);
    }

    float horizontal = Input.GetAxis("Horizontal");
    if (speed <= 200.0f) { speed += 1f; }

    if (speed > 65)
    {
      myRigidbody.velocity = new Vector2(direction * speed, myRigidbody.velocity.y);
    }
  }

  private void CheckCollider(Collider2D col)
  {
    switch(elgarState)
    {
      case ElgarState.CHARGE:
        CheckChargeStopZones(col);
        break;
    }
  }

 private void CheckChargeStopZones(Collider2D col)
  {
		gameObject.GetComponent<AudioSource> ().Play ();//stop run sound
    if (col.CompareTag("StopZone1"))
    {
```

```
                    gameObject.GetComponent<AudioSource> ().Stop ();
        Camera.main.GetComponent<CameraEffects>().Shake(0.2f, 0.3f, 0.6f);
        col.GetComponent<AudioSource> ().Play ();
        ChangeDirection(0.1f);
        chargesCount++;
    }
    else if (col.CompareTag("StopZone2"))
    {
                    gameObject.GetComponent<AudioSource> ().Stop ();
        Camera.main.GetComponent<CameraEffects>().Shake(0.2f, 0.3f, 0.6f);
        col.GetComponent<AudioSource> ().Play ();
        ChangeDirection(-0.1f);
        chargesCount++;
    }
}

private void ChangeDirection(float dir)
{
    if(!hasChangedDirection)
    {
        hasChangedDirection = true;
        myAnimator.SetBool("Stoped", true);
        if (dir < 0) transform.rotation = Quaternion.Euler(Vector3.zero);
        else transform.rotation = Quaternion.Euler(Vector3.up * 180);
        speed = 0;
        direction = dir;
    }
}

private IEnumerator SetNextState(ElgarState nextState)
{
                    myAnimator.SetBool("Bouncing", false);
                    myAnimator.SetBool("Stalactite", false);
    myAnimator.SetBool("Stoped", true);
    myRigidbody.velocity = new Vector2(0, 0);

    //If we are in bouncing state, modify physics material
    if(elgarState == ElgarState.BOUNCE)
    {
        GetComponent<CircleCollider2D>().sharedMaterial = stopBouncingMaterial;
        yield return new WaitForSeconds(1.5f);
    }
    vulnerableTrigger.SetActive(true);

    yield return new WaitForSeconds(1.0f);
    InitialiseElgarToState(nextState);
}

private void ScalactiteFall(){
                    myAnimator.SetBool("Stalactite", true);
    myRigidbody.AddForce(Vector2.up * 15, ForceMode2D.Impulse);
    Invoke("FreezeYOnScalactites", 0.6f);
    Invoke("ReturnToGround", 2.50f);
                    //efecto salto inicio
                    soundShout.GetComponent<AudioSource>().Play();

        }

private void FreezeYOnScalactites()
{
    myRigidbody.constraints = RigidbodyConstraints2D.FreezePositionY | RigidbodyConstraints2D.FreezeRotation;
}


private void ReturnToGround()
{
    myRigidbody.AddForce(-Vector2.up * 25, ForceMode2D.Impulse);
    myRigidbody.constraints = RigidbodyConstraints2D.FreezeRotation;
    Camera.main.GetComponent<CameraEffects>().Shake(1.0f, 0.7f, 0.35f);
    Invoke("StalactitePiecesFallPhysics", 0.5f);
}
```

```
    private void StalactitePiecesFallPhysics()
    {
                objEscalactita1.GetComponent<AudioSource> ().Play ();
        objEscalactita1.transform.position = new Vector2(UnityEngine.Random.Range(-10f, -5f), 12.5f);//9.15f,39f),12.5f);
        objEscalactita1.GetComponent<Rigidbody2D>().gravityScale = UnityEngine.Random.Range(0.5f, 3f);
        objEscalactita2.transform.position = new Vector2(UnityEngine.Random.Range(-4f, 1f), 12.5f);//9.15f,39f),12.5f);
        objEscalactita2.GetComponent<Rigidbody2D>().gravityScale = UnityEngine.Random.Range(0.5f, 3f);
        objEscalactita3.transform.position = new Vector2(UnityEngine.Random.Range(2f, 7f), 12.5f);//9.15f,39f),12.5f);
        objEscalactita3.GetComponent<Rigidbody2D>().gravityScale = UnityEngine.Random.Range(0.5f, 3f);
                objEscalactita4.transform.position = new Vector2(UnityEngine.Random.Range(8f, 14f), 12.5f);//9.15f,-
39f),12.5f);
                objEscalactita4.GetComponent<Rigidbody2D>().gravityScale = UnityEngine.Random.Range(0.5f, 3f);
                objEscalactita5.transform.position = new Vector2(UnityEngine.Random.Range(15f, 20f), 12.5f);//9.15f,-
39f),12.5f);
                objEscalactita5.GetComponent<Rigidbody2D>().gravityScale = UnityEngine.Random.Range(0.5f, 3f);
        Invoke("IncrementStalactitesCount", 1.0f);
    }

    private void IncrementStalactitesCount()
    {
        stalactitesCount++;
    }
}
```

CollisionTrigger.cs

```
using UnityEngine;
using System.Collections;

public class CollisionTrigger : MonoBehaviour {

        private BoxCollider2D playerCollider;

        [SerializeField]
        private BoxCollider2D platformCollider;
        [SerializeField]
        private BoxCollider2D platformTrigger;

        // Use this for initialization
        void Start () {
                playerCollider = GameObject.FindGameObjectWithTag("Player").GetComponent<BoxCollider2D> ();
                Physics2D.IgnoreCollision (platformCollider, platformTrigger, true);
        }

        void OnTriggerEnter2D(Collider2D other)
        {
                if (other.gameObject.tag == "Player") {
                        Physics2D.IgnoreCollision (platformCollider, playerCollider, true);
                }
        }
        void OnTriggerExit2D(Collider2D other)
        {
                if (other.gameObject.tag == "Player") {
                        Physics2D.IgnoreCollision (platformCollider, playerCollider, false);
                }
        }

}
```

controls.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class controls : MonoBehaviour {
	public GameObject musicPlayerSFX;
	public Button boton_back_controls;
	// Use this for initialization
	void Start () {
		musicPlayerSFX = GameObject.FindGameObjectWithTag (“SfxPushBack”);
	}

	public void ReturnMenu(){
		musicPlayerSFX.GetComponent<AudioSource> ().Play ();
		SceneManager.LoadScene(“main_menu”);
	}

}
```

credits.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class credits : MonoBehaviour {
	public GameObject musicPlayerSFX;
	public Button boton_back_credits;
	public GameObject mainMusic;
	// Use this for initialization
	void Start () {
		musicPlayerSFX = GameObject.FindGameObjectWithTag (“SfxPushBack”);
		mainMusic = GameObject.FindGameObjectWithTag (“MainMenuTheme”);
		Destroy (mainMusic);
		//Destroy(musicPlayer);
	}
	public void ReturnMenu(){
		musicPlayerSFX.GetComponent<AudioSource> ().Play ();
		SceneManager.LoadScene(“main_menu”);
	}

}
```

ElgarVulnerableTrigger.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

[RequireComponent(typeof(RenderEffects))]
public class ElgarVulnerableTrigger : MonoBehaviour {
   // definir tiempo cd entre golpes
   public float maxHitTime = 1.5f;
   private float hitTime;

	private AudioSource sfxHit, musicBoss, musicBossKill;
	private Image content;
	private GameObject player, elgar, postBossDialog;
   private RenderEffects effects;
```

```
void OnTriggerEnter2D(Collider2D col)
  {
    if(col.CompareTag("Player") && (Time.time - hitTime) > maxHitTime)
    {
                        if (player.GetComponent<Animator> ().GetBool ("land")) {
                                player.GetComponent<Player> ().invulnerable = true;
                                StartCoroutine (Invulnerability (0.3f));
                                hitTime = Time.time;
                                Debug.Log ("[Elgar] GOT HIT BY PLAYER! 1 LIFE LESS ON BOSS !! lets lower his HP points
by 10");
                                content.fillAmount = content.fillAmount - 0.1f;
                                if (content.fillAmount > 0f) {
                                        ////myRigidbody.AddForce (new Vector2 (0, jumpForce));player.GetCompo-
nent<Rigidbody2D>()
                                        player.GetComponent<Rigidbody2D> ().velocity = new Vector2 (0, 0);
                                        player.GetComponent<Rigidbody2D> ().AddForce (new Vector2 (0, 450));
                                        sfxHit.Play ();

                                        effects.Blink (
                                                transform.parent.GetComponent<SpriteRenderer> (),
                                                Color.red,
                                                1.0f);
                                } else {//GAME OVER WIN
              //player.transform.position = new Vector2 (-8.5f, -4);
            StartCoroutine(EndingBattle(player.transform, new Vector2(-8.5f, -4f)));
                                        player.GetComponent<Player> ().inBossStage = true;
                                        player.GetComponent<Animator> ().SetFloat ("speed", 0);
                                        //player.GetComponent<Animator> ().speed = 0;
                                        player.GetComponent<Animator> ().SetBool ("land", true);
                                        player.GetComponent<Rigidbody2D> ().velocity = new Vector2 (0, 0);
                                        //
                                        StartCoroutine(EndingBattle(elgar.transform, new Vector2(12f, -4f)));
                                        //elgar.transform.position = new Vector2 (12, -4);//Vector2.Lerp(elgar.Get-
Component<Rigidbody2D> ().position,new Vector2(0f,0f),3);//
                                        elgar.GetComponent<CharacterElgar> ().bossBegin = false;
                                        elgar.GetComponent<Animator> ().SetBool ("Stoped", true);
                                        elgar.GetComponent<Rigidbody2D> ().velocity = new Vector2 (0, 0);
                                        //
                                        musicBoss.Stop ();
                                        musicBossKill.Play ();
                                        //
                                        //postBossDialog.SetActive(true);
                                        postBossDialog.GetComponent<Animator> ().enabled = true;
                                        elgar.GetComponent<AudioSource>().Stop();
                                        //postBossDialog.GetComponent<Animator> ().Play ();
                                }
                        }

    }
  }

  private IEnumerator EndingBattle(Transform element, Vector2 endPosition)
  {
    float time = 0.0f;
    float maxTime = 2.0f;

    Vector2 initialPosition = element.position;
    while (time < 1)
    {
      time += Time.deltaTime / maxTime;
      element.position = Vector2.Lerp(initialPosition, endPosition, time);
      yield return null;
    }
  }

          IEnumerator Invulnerability(float howmanyseconds){
                  yield return new WaitForSeconds (howmanyseconds);
                  player.GetComponent<Player> ().invulnerable = false;
          }
}
```

enemy.cs

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

public class enemy : MonoBehaviour {
	private Rigidbody2D myRigidbody;
	private Animator myAnimator;

	[SerializeField]
	private float monsterSpeed;
	[SerializeField]
	private bool startToLeft;
	// Use this for initialization
	private int monsterDirection;

	void Start () {
		if (startToLeft == true) {
			monsterDirection = -1;
		} else {
			monsterDirection = 1;
		}
		myAnimator = this.GetComponent<Animator> ();
		myRigidbody = this.GetComponent<Rigidbody2D> ();
		//myRigidbody = gameObject.GetComponent<Rigidbody2D>;
		//myRigidbody.velocity = new Vector2(5, myRigidbody.velocity.y);
	}

	// Update is called once per frame
	void Update () {
		//myRigidbody = new Vector2 (50, 0);
		if (!myAnimator.GetBool ("death")) {
			myRigidbody.velocity = new Vector2 (monsterSpeed * monsterDirection, 0);
		} else {
			myRigidbody.velocity = new Vector2 (0, 0);
		}
		//myRigidbody.AddForce(new Vector2(25, 0), ForceMode2D.Impulse);
	}

	void OnTriggerEnter2D(Collider2D col)
	{
		if (col.CompareTag ("Babosa")) {
			if (monsterDirection == 1) {
				gameObject.GetComponent<SpriteRenderer> ().flipX = false;
				monsterDirection = -1;
			} else {
				gameObject.GetComponent<SpriteRenderer> ().flipX = true;
				monsterDirection = 1;
			}
		}
		if (col.CompareTag("StopZone1")) {
			gameObject.GetComponent<SpriteRenderer> ().flipX = true;
			monsterDirection = 1;
		}
		if(col.CompareTag("StopZone2")) {
			gameObject.GetComponent<SpriteRenderer> ().flipX = false;
			monsterDirection = -1;
		}
		/*
		if (col.CompareTag ("Player")) {
			if (col.GetComponent<Animator> ().GetBool ("land") == true) {
				Debug.Log ("HIT!!!!!!!");
			}
		}
		*/
	}
}
```

fading.cs

```
using UnityEngine;
using System.Collections;

public class fading : MonoBehaviour {

	public Texture2D fadeOutTexture;
	public float fadeSpeed = 0.8f;

	private int drawDepth = -1000;
	private float alpha = 1.0f;
	private int fadeDir = -1;

	void OnGUI (){
		alpha += fadeDir * fadeSpeed * Time.deltaTime;
		alpha = Mathf.Clamp01 (alpha);
		GUI.color = new Color (GUI.color.r, GUI.color.g, GUI.color.b, alpha);
		GUI.depth = drawDepth;
		GUI.DrawTexture ( new Rect (0, 0, Screen.width, Screen.height), fadeOutTexture );
	}
	public float BeginFade (int direction) {
		fadeDir = direction;
		return (fadeSpeed);
	}
	/*
	void OnLevelWasLoaded (){
		BeginFade (-1);
	}
	*/

}
```

GameController.cs

```
using UnityEngine;
using System.Collections;

public class GameController : MonoBehaviour {

	public GUIText scoreText;
	public int score;

	void Start ()
	{
		score = 0;
		UpdateScore ();
	}
	public void AddScore (int newScoreValue)
	{
		score += newScoreValue;
		UpdateScore ();
	}
	void UpdateScore()
	{
		scoreText.text = “score: “ + score;
	}
}
```

GameOverWin.cs

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

public class GameOverWin : MonoBehaviour {
	private AudioSource musicending;
	private GameObject gameover,elgarHpBar;
	// Use this for initialization
	void Start () {
		elgarHpBar = GameObject.FindWithTag("ElgarCanvas");
		musicending = GameObject.FindWithTag ("MusicEnding").GetComponent<AudioSource> ();
		gameover = GameObject.FindWithTag ("GameOver");
		//gameObject.GetComponent<Animator> ().Stop ();
		gameObject.GetComponent<Animator>().enabled = false;

	}

	private void PlayEndingMusic()
	{
		elgarHpBar.GetComponent<CanvasGroup>().alpha = 0;
		musicending.Play ();
	}

	private void StartEndingDialog(){
		gameObject.SetActive (true);
	}
	private void ShowGameOver(){
		//gameover.SetActive (true);

		gameObject.GetComponent<Animator> ().Stop ();
	}

}
```

introIth.cs

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

public class introIth : MonoBehaviour {
	private Animator myAnimator;
	private GameObject player;
	private GameObject dialog;
	private Camera myCamera;
   private PlayerState playerState;
   private Transform startPoint;

	// Use this for initialization
	void Start () {
		myAnimator = gameObject.GetComponent<Animator> ();
		player = GameObject.FindGameObjectWithTag ("Player");
		dialog = GameObject.FindGameObjectWithTag ("DialogIntroITH");
     playerState = GameObject.FindGameObjectWithTag("PlayerState").GetComponent<PlayerState>();
     startPoint = GameObject.Find("StartPoint").GetComponent<Transform>();

     dialog.SetActive(false);

		myCamera = Camera.main;
	}

	// Update is called once per frame
	void Update () {
		if (!playerState.hasSeenScene1 && this.gameObject.transform.position.y <= 0) {
			myAnimator.SetBool ("eggland", true);
			StartCoroutine (KillEgg(1.8f));
		}
```

```
    else if(playerState.hasSeenScene1)
      {
        RelocatePlayerAndRemoveEgg(startPoint.position);
      }
            }
            IEnumerator KillEgg(float howmanyseconds){
                        //player.GetComponent<Player>().controlEnabled = false;
                        //player.GetComponent<Player>().inBossStage = true;
                        yield return new WaitForSeconds (howmanyseconds);
      RelocatePlayerAndRemoveEgg(new Vector2(gameObject.transform.position.x, gameObject.transform.position.y));
      dialog.SetActive(true);
                        //player.GetComponent<Player>().controlEnabled = true;
                        //player.GetComponent<Player>().inBossStage = false;

      playerState.hasSeenScene1 = true;
    }

    private void RelocatePlayerAndRemoveEgg(Vector2 initialPosition)
    {
      player.transform.position = initialPosition;
      Camera.main.GetComponent<CameraFollow>().target = player.transform;
      this.gameObject.SetActive(false);
    }
}
```

introMovie.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;


[RequireComponent (typeof(AudioSource))]
public class introMovie : MonoBehaviour {
            public MovieTexture movie;
            public AudioSource audio;
            private bool manualpause;
            private Texture myRaw;
            private GameObject skiptext;
            private bool waitload;
            /*
            void Awake(){
                        myRaw=GetComponent<RawImage> ().texture = movie as MovieTexture;
            }
            */
            void Start () {
                        waitload = false;
                        manualpause = false;
                        skiptext = GameObject.FindGameObjectWithTag (“SkipText”);
                        GetComponent<RawImage> ().texture = movie as MovieTexture;
                        //GetComponent<RawImage>().material.mainTexture=movie;
                        audio = GetComponent<AudioSource> ();
                        audio.clip = movie.audioClip;
                        movie.Play ();
                        audio.Play ();
                        //loadLevel1();
            }

            void Update () {
                        if (!waitload) {
                                    if (Input.GetKeyDown (KeyCode.Space) && movie.isPlaying) {
                                                manualpause = true;
                                                movie.Pause ();
                                    } else if (Input.GetKeyDown (KeyCode.Space) && !movie.isPlaying) {
                                                manualpause = false;
                                                movie.Play ();
                                    }
```

```
                        if (!movie.isPlaying && !manualpause) {
                                Debug.Log (“loadScene1”);
                                loadLevel1 ();
                        }
                        if (Input.GetKeyDown (KeyCode.Escape)) {
                                loadLevel1 ();
                        }
                }

        }
        private void loadLevel1(){
                manualpause = true;
                movie.Pause ();
                audio.Pause ();
                waitload = true;
                skiptext.SetActive (false);
                SceneManager.LoadScene(“level_01”);
        }
}
```

Jumper.cs

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

public class Jumper : MonoBehaviour {
        [SerializeField]
        private float fuerzaX;
        [SerializeField]
        private float fuerzaY;

        void OnTriggerEnter2D(Collider2D col){
                GameObject.FindWithTag (“Player”).GetComponent<Rigidbody2D> ().velocity = new Vector3(0,0,0);
                GameObject.FindWithTag (“Player”).GetComponent<Rigidbody2D>().AddForce (new Vector2 (fuerzaX,
fuerzaY), ForceMode2D.Impulse);
                //GameObject.FindWithTag (“Jumper1”).GetComponent<AudioSource> ().Play ();
                gameObject.GetComponent<AudioSource>().Play();
                gameObject.GetComponent<Animation> ().Play ();
        }
}
```

LevelHandler.cs

```
using UnityEngine;
using System.Collections;
using System;
using UnityEngine.SceneManagement;

public class LevelHandler : MonoBehaviour {
        private static LevelHandler instance;

        public GUITexture overlay;
        public float fadeTime;

        public static LevelHandler Instance
        {
                get {
                        if (instance == null) {
                                instance = GameObject.FindObjectOfType<LevelHandler> ();
                        }
                        return instance;
                }

        }
```

```
	void Awake()
	{
		overlay.pixelInset = new Rect (0, 0, Screen.width, Screen.height);
		StartCoroutine (FadeToClear ());

	}
	public void ReloadLevel()
	{
		//StartCoroutine(FadeToBlack(() => SceneManager.LoadScene(SceneManager.GetActiveScene())));
		SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex);
	}
	private IEnumerator FadeToClear()
	{
		overlay.gameObject.SetActive (true);
		overlay.color = Color.black;
		float rate = 1.0f / fadeTime;
		float progress = 0.0f;
		while (progress < 1.0f)
		{
			overlay.color = Color.Lerp (Color.black, Color.clear, progress);
			progress += rate * Time.deltaTime;
			yield return null;
		}
		overlay.color = Color.clear;
		overlay.gameObject.SetActive (false);
	}

	private IEnumerator FadeToBlack(Action levelMethod)
	{
		overlay.gameObject.SetActive (true);
		overlay.color = Color.clear;
		float rate = 1.0f / fadeTime;
		float progress = 0.0f;
		while (progress < 1.0f)
		{
			overlay.color = Color.Lerp (Color.clear, Color.black, progress);
			progress += rate * Time.deltaTime;
			yield return null;
		}
		overlay.color = Color.black;
		levelMethod ();
	}
}
```

main_menu.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class main_menu : MonoBehaviour {

	public Button boton_play;
	public Button boton_controls;
	public Button boton_credits;
	public Button boton_exit;

	// Use this for initialization
	public GameObject musicPlayer;

	void Start () {
		musicPlayer=GameObject.Find ("MUSIC");
	}
	public void PlayGame(){
		GameObject.FindWithTag ("SfxPush").GetComponent<AudioSource> ().Play ();
		Destroy(musicPlayer);
		SceneManager.LoadScene("intro_movie");//level_01
	}
```

```
	public void OpControls(){
		GameObject.FindWithTag (“SfxPush”).GetComponent<AudioSource> ().Play ();
		SceneManager.LoadScene(“controls”);
	}
	public void OpCredits(){
		GameObject.FindWithTag (“SfxPush”).GetComponent<AudioSource> ().Play ();
		SceneManager.LoadScene(“credits”);
	}
	public void OpQuit(){
		Application.Quit ();
	}
}
```

menu_music.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class menu_music : MonoBehaviour {
	public GameObject musicPlayer;
	public GameObject sfxPushButton;

	void Awake (){

		musicPlayer = GameObject.Find (“MUSIC”);
		//sfxPushButton = GameObject.Find (“SFXpush”);

		if (musicPlayer == null) {
			musicPlayer = this.gameObject;
			musicPlayer.name = “MUSIC”;
			DontDestroyOnLoad (musicPlayer);
		} else {
			if (this.gameObject.name != “MUSIC”) {
				Destroy (this.gameObject);
			}
		}

		/*
		if (sfxPushButton == null) {
			sfxPushButton = this.gameObject;
			sfxPushButton.name = “SFXpush”;
			DontDestroyOnLoad (sfxPushButton);
		}
		*/
		//DontDestroyOnLoad(this.gameObject);
		//DontDestroyOnLoad(transform.gameObject);
	}

}
```

PauseMenu.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour {

	public GameObject PauseUI;
	private bool paused = false;
	private PlayerState playerState;
	void Start(){
		playerState = GameObject.FindGameObjectWithTag(“PlayerState”).GetComponent<PlayerState>();

		PauseUI.SetActive (false);
	}
```

```
	void Update(){
		//if(Input.GetButtonDown(*Pause*)){
		if (Input.GetKeyDown (KeyCode.Escape)) {
			if (!paused) {
				GameObject.FindWithTag (“PauseSound”).GetComponent<AudioSource> ().Play ();
			}
			GameObject.FindWithTag (“GameMusic”).GetComponent<AudioSource> ().UnPause ();
			paused = !paused;
		}

		if(paused){
			PauseUI.SetActive(true);
			AudioListener.pause = true;
			GameObject.FindWithTag (“GameMusic”).GetComponent<AudioSource> ().Pause ();
			Time.timeScale=0;
		}

		if(!paused){
			AudioListener.pause = false;
			PauseUI.SetActive(false);
			Time.timeScale=1;
		}

	}

	public void Resume(){
		GameObject.FindWithTag (“GameMusic”).GetComponent<AudioSource> ().UnPause ();
		paused = false;
	}

	public void Restart(){
		playerState.Reset();
		SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex);
	}

	public void MainMenu(){
		paused = false;
		SceneManager.LoadScene(“main_menu”);
	}
	public void Quit(){
		Application.Quit ();
	}

}
```

platformLeave.cs

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

public class platformLeave : MonoBehaviour {
	private GameObject player;
	private Transform childTransform;
	// Use this for initialization
	void Start () {
		player=GameObject.FindGameObjectWithTag(“Player”);
	}

	void OnTriggerEnter2D(Collider2D col)
	{
		if(col.CompareTag(“Player”)){
			player.transform.SetParent (null);
		}
	}
}
```

PlatformMovement.cs

```
using UnityEngine;
using System.Collections;

public class PlatformMovement : MonoBehaviour {

	private Vector2 posA;
	private Vector2 posB;
	private Vector2 nexPos;
	private GameObject player;

	[SerializeField]
	private float speed;

	[SerializeField]
	private Transform childTransform;

	[SerializeField]
	private Transform transformB;
	// Use this for initialization
	private bool incolision;
	void Start () {
		player=GameObject.FindGameObjectWithTag("Player");
		incolision = false;
		posA = childTransform.localPosition;
		posB = transformB.localPosition;
		nexPos = posB;
	}

// Update is called once per frame
void FixedUpdate(){
		Move ();
		if (incolision) {
			Debug.Log ("incolision");
			if (player.transform.position.x < posA.x || player.transform.position.x > posB.x) {
				Debug.Log ("delete colision");
				player.transform.SetParent (null);
				incolision = false;
			}
		}
	}
	private void Move()
	{
			childTransform.localPosition = Vector2.MoveTowards (childTransform.localPosition, nexPos,
speed * Time.deltaTime);
		if (Vector2.Distance (childTransform.localPosition, nexPos) <= 0.1)
		{
			ChangeDestination ();
		}
	}
	private void ChangeDestination()
	{
		nexPos = nexPos != posA ? posA : posB;
	}
}
```

Player.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityEngine.SceneManagement;
using System;

[RequireComponent(typeof(RenderEffects))]
public class Player : MonoBehaviour {
	private enum ElgarState { CHARGE, STALACTITE, BOUNCE };
	private Rigidbody2D myRigidbody;
```

```
private float posicionPersonajeX;
        private float posicionY;

        private Animator myAnimator;
    private RenderEffects effects;
        [SerializeField]
        public bool invulnerable;

        [SerializeField]
        private float timeInvulnerableAfterHit;

    [SerializeField]
    public bool inBossStage;

    public bool controlEnabled;

    [SerializeField]
        private float movementSpeed;

        private bool facingRight;

        [SerializeField]
        private Transform[] groundPoints;

        [SerializeField]
        private LayerMask whatIsGround;

        private bool isGrounded;

        private bool jump;

        [SerializeField]
        private bool airControl;

        [SerializeField]
        private float jumpForce;

        [SerializeField]
        private float groundRadius;


        //
        //private int countMana;
        private int countEgg;
        public Text countManaText;
        public Text countEggText;
        private GameObject txtRef, personaje;
        //

        //
        private AudioSource coinSoundEffect,sfxGameOver;//public
        private AudioSource sfxHit;

        private ParticleSystem ps;

        private float animaSpeed;
        //private float baseJumpForce;
        private float baseRunForce;
        private int puzzlecount;

        private Image content;
        //public Vector3 starPosition;


        private bool gameover;
        private Animator elgarBoss;
        private float damageAmount ;

    private PlayerState playerState;
```

```
        void Start () {
    damageAmount = 0.15f;
                gameover = false;
                GameObject personaje = GameObject.FindGameObjectWithTag(“Player”);
                //timeInvulnerableAfterHit = 0f;
    effects = GetComponent<RenderEffects>();
    puzzlecount = 0;
                animaSpeed=2f;
                baseRunForce = movementSpeed;
                ps = GameObject.FindGameObjectWithTag (“PauseSound”).GetComponentInChildren <ParticleSystem>
();
                sfxHit=GameObject.FindGameObjectWithTag(“SFXPlayerDamaged”).GetComponent<AudioSource>();
                sfxGameOver=GameObject.FindGameObjectWithTag(“SFXGameOver”).GetComponent<AudioSource>();

                content = GameObject.FindGameObjectWithTag (“IthHpBar”).GetComponent<Image> ();
                var emission = ps.emission;
                emission = ps.emission;
                controlEnabled = true;
                countEgg = 0;
                StartCoroutine(SetCountText(0f));
                facingRight = true;
                myRigidbody = GetComponent<Rigidbody2D> ();
                myAnimator = GetComponent<Animator> ();
    //Disable ith movement in boss (Start)
    /*if (inBossStage) {
      gameObject.GetComponent<Rigidbody2D>().constraints = RigidbodyConstraints2D.FreezePositionX;
      gameObject.GetComponent<Animator>().enabled = false;
    }*/
    InitialisePlayerState();

    if (inBossStage) {
                    elgarBoss = GameObject.FindGameObjectWithTag (“Elgar”).GetComponent<Animator>();
                }
  }

  private void InitialisePlayerState()
  {
    playerState = GameObject.FindGameObjectWithTag(“PlayerState”).GetComponent<PlayerState>();
    content.fillAmount = playerState.life;
    countManaText.text = playerState.mana.ToString();
  }

  void Update()
        {
                if (!inBossStage && !gameover)
    {
      HandleInput();
    }
    CheckGameOver();
  }
        // Update is called once per frame
        void FixedUpdate () {
                if (!inBossStage && !gameover)
    {
      //MOVEMENT SPEED AND JUMP INCREASED WITH MANA STARS
      movementSpeed = baseRunForce + (playerState.mana / 40);

      float horizontal = Input.GetAxis(“Horizontal”);

      isGrounded = IsGrounded();
      //Debug.Log (horizontal);
      HandleMovement(horizontal);

      Flip(horizontal);

      HandleLayers();

      ResetValues();
    }
```

```
        if(gameover){
                Time.timeScale=0;
            }
        }

        private void HandleMovement(float horizontal)
        {
            if (myRigidbody.velocity.y < 0) {
                myAnimator.SetBool ("land", true);
            }
            //myRigidbody.velocity = Vector2.left;//x = -1, y=0
            /*
            if (isGrounded && airControl) {// ||
                myRigidbody.velocity = new Vector2 (horizontal * movementSpeed, myRigidbody.velocity.y);
            } else if (isGrounded) {
                myRigidbody.velocity = new Vector2 (horizontal * movementSpeed, myRigidbody.velocity.y);
            } else if (myAnimator.GetBool ("land")==false && !isGrounded) {
                myRigidbody.velocity = new Vector2 (horizontal * movementSpeed, myRigidbody.velocity.y);
            } else if (myAnimator.GetBool ("land")==true){//(!isGrounded) {
                myRigidbody.velocity = new Vector2 (horizontal * movementSpeed / 1.5f, myRigidbody.velocity.y);
            }*/
            myRigidbody.velocity = new Vector2 (horizontal * movementSpeed, myRigidbody.velocity.y);
            myAnimator.SetFloat ("speed", Mathf.Abs(horizontal));

            if (isGrounded && jump && !myAnimator.GetBool("land"))
            {
                    GetComponent<AudioSource> ().Play ();
                    isGrounded = false;
                    myRigidbody.AddForce (new Vector2 (0, jumpForce));
                    myAnimator.SetTrigger ("jump");
            }
            if (myAnimator.GetBool ("land")==true && isGrounded && !GameObject.FindWithTag ("SoundLanding").GetComponent<AudioSource> ().isPlaying) {
                GameObject.FindWithTag ("SoundLanding").GetComponent<AudioSource> ().Play ();
            }

        }

        private void HandleInput()
        {
            if (Input.GetKeyDown (KeyCode.Space)) {
                    jump = true;
            }
        }

        private void Flip(float horizontal)
        {
            if (horizontal > 0 && !facingRight || horizontal < 0 && facingRight)
            {
                facingRight = !facingRight;
                Vector3 theScale = transform.localScale;
                theScale.x *= -1;
                transform.localScale = theScale;
            }
        }

        private bool IsGrounded()
        {
            if (myRigidbody.velocity.y <= 0)
            {
                foreach (Transform point in groundPoints)
                {
                    Collider2D[] colliders = Physics2D.OverlapCircleAll (point.position, groundRadius, whatIsGround);

                    for (int i = 0; i < colliders.Length; i++) {
                        if (colliders [i].gameObject != gameObject) {
                            myAnimator.ResetTrigger ("jump");
                            myAnimator.SetBool ("land", false);
                            return true;
                        }
```

```
                                        }
                                }
                        }
                        return false;
                }

                private void ResetValues()
                {
                        jump = false;
                }
                private void HandleLayers()
                {
                        if (!isGrounded) {
                                myAnimator.SetLayerWeight (1, 1);
                        } else {
                                myAnimator.SetLayerWeight (1, 0);
                        }
                }
/////
//private Sprite imgPuzzle1;
void OnTriggerExit2D(Collider2D col)
{
   if(col.CompareTag("MovingPlatform"))
   {
      //Remove Ith from platform
      transform.parent = null;
   }
}

void OnTriggerEnter2D(Collider2D col)
        {
   //MANA COLLISION
   if (col.CompareTag("Mana"))
   {
                                col.gameObject.GetComponentInChildren<BoxCollider2D> ().enabled=false;
      playerState.AddStars(1);
      StartCoroutine(SetCountText(0.5f));
                                col.gameObject.GetComponent<AudioSource> ().Play ();
      //WATER COLLISION
   }
   else if (col.CompareTag("BigMana"))
   {
                                col.gameObject.GetComponentInChildren<BoxCollider2D> ().enabled=false;
      playerState.AddStars(5);
      StartCoroutine(SetCountText(0.5f));
      GameObject.FindWithTag("BigManaItems").GetComponent<AudioSource>().Play();
   }
   else if (col.CompareTag("Water"))
   {
      //WATER PARTICLE EFFECT (MOVE TO PLAYER POSITION THEN PLAY ANIMATION)


      posicionPersonajeX = gameObject.transform.position.x;
      posicionY = -7f;
      GameObject.FindWithTag("WaterEffect").transform.position = new Vector3(posicionPersonajeX, posicionY);
      GameObject.FindWithTag("WaterEffect").GetComponent<ParticleSystem>().Play();
      GameObject.FindWithTag("Water").GetComponent<AudioSource>().Play();
      GameObject.FindWithTag("Water").GetComponent<BoxCollider2D>().enabled = false;
      controlEnabled = false;
      //RELOAD SCENE
      //StartCoroutine(reloadScene(2));
      playerState.Damage(1.0f);
   }

  else if (col.CompareTag("MovingPlatform"))
   {
                                gameObject.transform.SetParent (col.transform);
      //transform.parent = col.transform;
   }
```

```
        else if (col.CompareTag("Puzzle1"))
    {
                        col.gameObject.GetComponentInChildren<BoxCollider2D> ().enabled=false;
                        StartCoroutine (SetPuzzle (0.5f,1));
      GameObject.FindWithTag("EggItems").GetComponent<AudioSource>().Play();
                        puzzlecount = puzzlecount + 1;
    }
    else if (col.CompareTag("Puzzle2"))
    {
                        col.gameObject.GetComponentInChildren<BoxCollider2D> ().enabled=false;
                        StartCoroutine (SetPuzzle (0.5f,2));
                        GameObject.FindWithTag("EggItems").GetComponent<AudioSource>().Play();
                        puzzlecount = puzzlecount + 1;
    }
    else if (col.CompareTag("Puzzle3"))
    {
                        col.gameObject.GetComponentInChildren<BoxCollider2D> ().enabled=false;
                        StartCoroutine (SetPuzzle (0.5f,3));
                        GameObject.FindWithTag("EggItems").GetComponent<AudioSource>().Play();
                        puzzlecount = puzzlecount + 1;
    }
            else if (col.CompareTag("Enterboss"))
            {
                        col.GetComponent<AudioSource> ().Play ();
                        StartCoroutine(loadBossScene(1));
            }

//ESCALACTITAS
            else if (col.CompareTag("Escalactita1") || col.CompareTag("Escalactita2") || col.CompareTag("Escalacti-
ta3")|| col.CompareTag("Escalactita4")|| col.CompareTag("Escalactita5") )
            {
                        if (!invulnerable) {
                                    invulnerable = true;
                                    StartCoroutine(Invulnerability (timeInvulnerableAfterHit));

                                    Debug.Log ("HIT!!!!!!!!!!!!");
                                    sfxHit.Play ();
        playerState.Damage(0.20f);
                                    if (content.fillAmount > 0) {
          content.fillAmount = playerState.life;
        }
                                    effects.Blink (
                                                GetComponent<SpriteRenderer> (),
                                                Color.red,
                                                1.0f);
                        }

} else if (col.CompareTag ("Babosa")) {
                        if (!invulnerable && !gameObject.GetComponent<Animator> ().GetBool ("land") && !col.Get-
ComponentInParent<Animator>().GetBool("death")) {
                                    invulnerable = true;
                                    StartCoroutine (Invulnerability (timeInvulnerableAfterHit));
                                    sfxHit.Play ();
                                    Debug.Log ("Babosa hits");
        playerState.Damage(0.10f);
        if (content.fillAmount > 0) {
          content.fillAmount = playerState.life;
                                    }
                                    effects.Blink (
                                                GetComponent<SpriteRenderer> (),
                                                Color.red,
                                                1.0f);
                        }
                        if(gameObject.GetComponent<Animator>().GetBool("land") && !col.GetComponentInParent<A-
nimator>().GetBool("death")){
                                    if (myRigidbody.velocity.y < jumpForce) {
                                                myRigidbody.AddForce (new Vector2 (0, jumpForce + 500));
                                    }
                                    col.GetComponentInParent<AudioSource> ().Play ();
                                    col.GetComponentInParent<Animator>().SetBool("death",true);
                                    invulnerable = true;
```

```
StartCoroutine (Invulnerability (timeInvulnerableAfterHit));
                                        StartCoroutine (KillBabosa(col.transform.root.gameObject));
                                }
                        //BOSS
                        } else if (col.CompareTag (“Enemy”)) {
                                if (!invulnerable && !gameObject.GetComponent<Animator> ().GetBool (“land”)) {
                                        invulnerable = true;
                                        StartCoroutine(Invulnerability (timeInvulnerableAfterHit));
                                        sfxHit.Play ();
                                        //CHARGE, STALACTITE, BOUNCE

                                        if (elgarBoss.GetBool (“Bouncing”)) {
                                                //Debug.Log (“bouncing.....”);
                                                //damageAmount = 0.10f;
                playerState.Damage(0.10f);

        }
        else if (elgarBoss.GetBool (“Stalactite”)) {
                                                //Debug.Log (“stala.....”);
                                                //damageAmount = 0.15f;
                playerState.Damage(0.15f);

        }
        else if (!elgarBoss.GetBool (“Stoped”)) {
                                                //Debug.Log (“run.....”);
                                                //damageAmount = 0.25f;
                playerState.Damage(0.25f);

        }

        Debug.Log (“HIT!!!!!!!!!!!!”);
                                        if (content.fillAmount > 0) {
                content.fillAmount = playerState.life;
                                        }
                                        effects.Blink (
                                                GetComponent<SpriteRenderer> (),
                                                Color.red,
                                                1.0f);
                                }
                        }
    /*//DIE
    CheckGameOver();*/
        }


  private void CheckGameOver()
  {
    if (!gameover)
    {
      if (playerState.life <= 0)
      {
        StartCoroutine(Invulnerability(2));
        gameover = true;
        inBossStage = true;
        gameObject.GetComponent<Rigidbody2D>().velocity = new Vector2(0, 0);
        gameObject.GetComponent<Animator>().SetFloat(“Speed”, 0);
        myAnimator.SetBool(“land”, true);
        jump = false;
        Debug.Log(“ITH DIES”);
        controlEnabled = false;
        sfxGameOver.Play();
        //personaje.GetComponent<BoxCollider2D> ().enabled = false;
        StartCoroutine(reloadScene(2));
      }
    }
  }
```

```
IEnumerator reloadScene(int howmanyseconds){
	Debug.Log("Reloading scene and resting playerState");
	playerState.Reset();
	yield return new WaitForSeconds (howmanyseconds);
			SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex);
		}
		IEnumerator SetCountText(float howmanyseconds){
			yield return new WaitForSeconds (howmanyseconds);
	//countManaText.text = countMana.ToString ();//+" x";
	countManaText.text = playerState.mana.ToString();
			var emission = ps.emission;
			emission.rateOverTime = playerState.mana;
		}
		IEnumerator SetPuzzle(float howmanyseconds, int quepuzzle){
			yield return new WaitForSeconds (howmanyseconds);
			GameObject.FindWithTag ("UIpuzzle" + quepuzzle).SetActive (false);
			GameObject.FindWithTag ("Puzzle" + quepuzzle + "b").SetActive (false);
			if (puzzlecount == 3) {
				GameObject.FindWithTag("Barrier").SetActive(false);
			}
		}
		IEnumerator loadBossScene(int howmanyseconds){
			yield return new WaitForSeconds (howmanyseconds);
			SceneManager.LoadScene("level_boss");
		}
		IEnumerator Invulnerability(float howmanyseconds){
			yield return new WaitForSeconds (howmanyseconds);
			invulnerable = false;
		}
		IEnumerator KillBabosa(GameObject quebabosa){
			yield return new WaitForSeconds (1.8f);
			Destroy (quebabosa);
		}
}
```

PlayerState.cs

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

public class PlayerState : MonoBehaviour {
   private static PlayerState _instance;
   public static PlayerState Instance { get { return _instance; } }
   public bool hasSeenScene1, hasSeenBossScene;
   public float life;
   public int mana;

   void Awake()
   {
      if (_instance != null && _instance != this)
      {
         Destroy(this.gameObject);
      }
      else
      {
         _instance = this;
         Reset();
         _instance.hasSeenScene1 = _instance.hasSeenBossScene = false;
         DontDestroyOnLoad(transform.gameObject);
      }
   }

   public void Reset()
   {
      _instance.life = 1.0f;
      _instance.mana = 0;
   }
```

```
    public void Damage(float amount)
    {
        _instance.life -= amount;
    }

    public void AddStars(int amount)
    {
        _instance.mana += amount;
    }
}
```

popITH.cs

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

public class popITH : MonoBehaviour {

        void OnTriggerEnter2D(Collider2D col){
                if(col.CompareTag("Player")){
                        Destroy (gameObject);
                }
        }
}
```

RenderEffects.cs

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

public class RenderEffects : MonoBehaviour {

        public void Blink(SpriteRenderer sprite, Color color, float time)
    {
        Debug.Log("Blinking!");
        StopCoroutine("BlinkSprite");
        StartCoroutine(BlinkSprite(sprite, color, time));
    }

    private IEnumerator BlinkSprite(SpriteRenderer sprite, Color color, float time)
    {
        int count = 0;

        for(float i = 0; i<time; i+=Time.deltaTime)
        {
            if(count % 6 == 0)
            {
                sprite.color = color;
            } else
            {
                sprite.color = Color.white;
            }
            count++;
            yield return null;
        }

        sprite.color = Color.white;
    }
}
```

ResetScalactitas.cs

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

public class ResetScalactitas : MonoBehaviour {

	void Update(){
		if (gameObject.transform.position.y < -50) {
			gameObject.GetComponent<Rigidbody2D> ().gravityScale = 0;
			gameObject.GetComponent<Rigidbody2D> ().velocity = new Vector2(0,0);
			gameObject.transform.position = new Vector2 (gameObject.transform.position.x, 12.5f);
		}
	}
}
```

ScrollingBackground.cs

```
using UnityEngine;
using System.Collections;

public class ScrollingBackground : MonoBehaviour {

	public bool scrolling,paralax;

	public float backgroundSize;
	public float paralaxSpeed;

	private Transform cameraTransform;
	private Transform[] layers;
	private float viewZone = 10;
	private int leftIndex;
	private int rightIndex;
	private float lastCameraX;

	private void Start()
	{
		cameraTransform = Camera.main.transform;
		lastCameraX = cameraTransform.position.x;
		layers = new Transform[transform.childCount];
		for (int i = 0; i < transform.childCount; i++)
			layers [i] = transform.GetChild (i);

		leftIndex = 0;
		rightIndex = layers.Length - 1;
	}

	private void Update()
	{
		if (paralax) {
			float deltaX = cameraTransform.position.x - lastCameraX;
			float deltaY = 5;
			transform.position += Vector3.right * (deltaX * paralaxSpeed);
			//transform.position.y = deltaY;
		}

		lastCameraX = cameraTransform.position.x;

		if (scrolling) {
			if (cameraTransform.position.x < (layers [leftIndex].transform.position.x + viewZone))
				ScrollLeft ();
			if (cameraTransform.position.x > (layers [rightIndex].transform.position.x - viewZone))
				ScrollRight ();
		}
	}
```

```
	private void ScrollLeft(){
		int lastRight = rightIndex;
		layers [rightIndex].position = Vector3.right * (layers [leftIndex].position.x - backgroundSize);
		leftIndex = rightIndex;
		rightIndex--;
		if (rightIndex < 0)
			rightIndex = layers.Length - 1;
	}
	private void ScrollRight(){
		int lastLeft = leftIndex;
		layers [leftIndex].position = Vector3.right * (layers [rightIndex].position.x + backgroundSize);
		rightIndex = leftIndex;
		leftIndex++;
		if (leftIndex == layers.Length)
			leftIndex = 0;
	}
}
```

sfx_click.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class sfx_click : MonoBehaviour {
	public AudioClip hover;
	public AudioClip click;
	public AudioSource audio_sfx;

	public void Onhover()
	{
		//audio_sfx.PlayOneShot (hover);
		//audio_sfx.PlayOneShot (hover);
		GameObject.FindWithTag ("SfxPush").GetComponent<AudioSource> ().Play ();
	}

	public void Onclick()
	{
		audio_sfx.PlayOneShot (click);
	}

	public GameObject musicPlayer;
	public GameObject sfxPushButton;

	void Start (){

		musicPlayer = GameObject.Find ("sfx_play");
		//sfxPushButton = GameObject.Find ("SFXpush");

		if (musicPlayer == null) {
			musicPlayer = this.gameObject;
			musicPlayer.name = "sfx_play";
			DontDestroyOnLoad (musicPlayer);
		} else {
			if (this.gameObject.name != "sfx_play") {
				Destroy (this.gameObject);
			}
		}
	}

}
```

SFX_playthis.cs

```
using UnityEngine;
using System.Collections;

public class SFX_playthis : MonoBehaviour {
	[SerializeField]
	public AudioSource audio_sfx;

	public void playSFX(){
		GameObject.FindWithTag (“SfxHover”).GetComponent<AudioSource> ().Play ();
		//audio_sfx.Play ();
	}
}
```

sfx_script.cs

```
using UnityEngine;
using System.Collections;

public class sfx_script : MonoBehaviour {
	public AudioSource audio_sfx;
	public GameObject musicPlayer;

	void Awake (){

		musicPlayer = GameObject.Find (“sfx_playback”);
		//sfxPushButton = GameObject.Find (“SFXpush”);

		if (musicPlayer == null) {
			musicPlayer = this.gameObject;
			musicPlayer.name = “sfx_playback”;
			DontDestroyOnLoad (musicPlayer);
		} else {
			if (this.gameObject.name != “sfx_playback”) {
				Destroy (this.gameObject);
			}
		}

	}
}
```

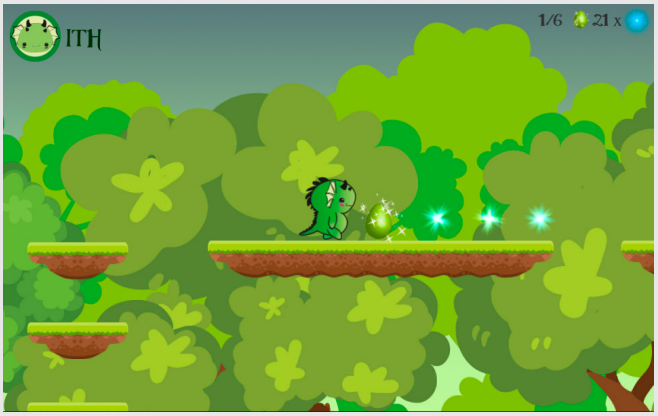
StartBoss.cs

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

public class StartBoss : MonoBehaviour {
   private GameObject player, bossMusic, elgar, elgarHpBar,prebossMusic,prebossIntro;
   private PlayerState playerState;
   [SerializeField]
   private bool skipIntro;


   void Start()
   {
      player = GameObject.FindWithTag(“Player”);
      elgar = GameObject.FindWithTag(“Elgar”);
      bossMusic = GameObject.FindWithTag(“MusicBoss”);
      elgarHpBar = GameObject.FindWithTag(“ElgarCanvas”);
      elgarHpBar.GetComponent<CanvasGroup>().alpha = 0;
      prebossMusic = GameObject.FindWithTag(“MusicPreBoss”);
```

```
        prebossIntro = GameObject.FindWithTag("PreBossIntro");
        playerState = GameObject.FindWithTag("PlayerState").GetComponent<PlayerState>();
        if (skipIntro || playerState.hasSeenBossScene) {
            Debug.Log("LOLOLOL : " + skipIntro + " | " + playerState.hasSeenBossScene);
            StartFight();
        }
    }

    private void StartFight()
    {
        playerState.hasSeenBossScene = true;
        gameObject.GetComponent<Animator>().enabled = false;

        //player.GetComponent<Rigidbody2D>().constraints = RigidbodyConstraints2D.None;
        //player.GetComponent<Animator>().enabled = true;
        prebossIntro.SetActive(false);
        player.GetComponent<Player>().inBossStage = false;
        bossMusic.GetComponent<AudioSource>().Play();
        prebossMusic.GetComponent<AudioSource>().Stop();
        elgar.GetComponent<CharacterElgar>().bossBegin = true;
        elgarHpBar.GetComponent<CanvasGroup>().alpha = 1;
    }

}
```

CAPTURAS DE PANTALLA

VERSIÓN ALPHA




REGIUS
PLAY
CONTROLS
CREDITS
EXIT
BETA VERSION 2.3


ITH
0
Where am I?
Brothers?...
I have to find them!


ITH
11


ITH
17


ITH
10


ITH
37

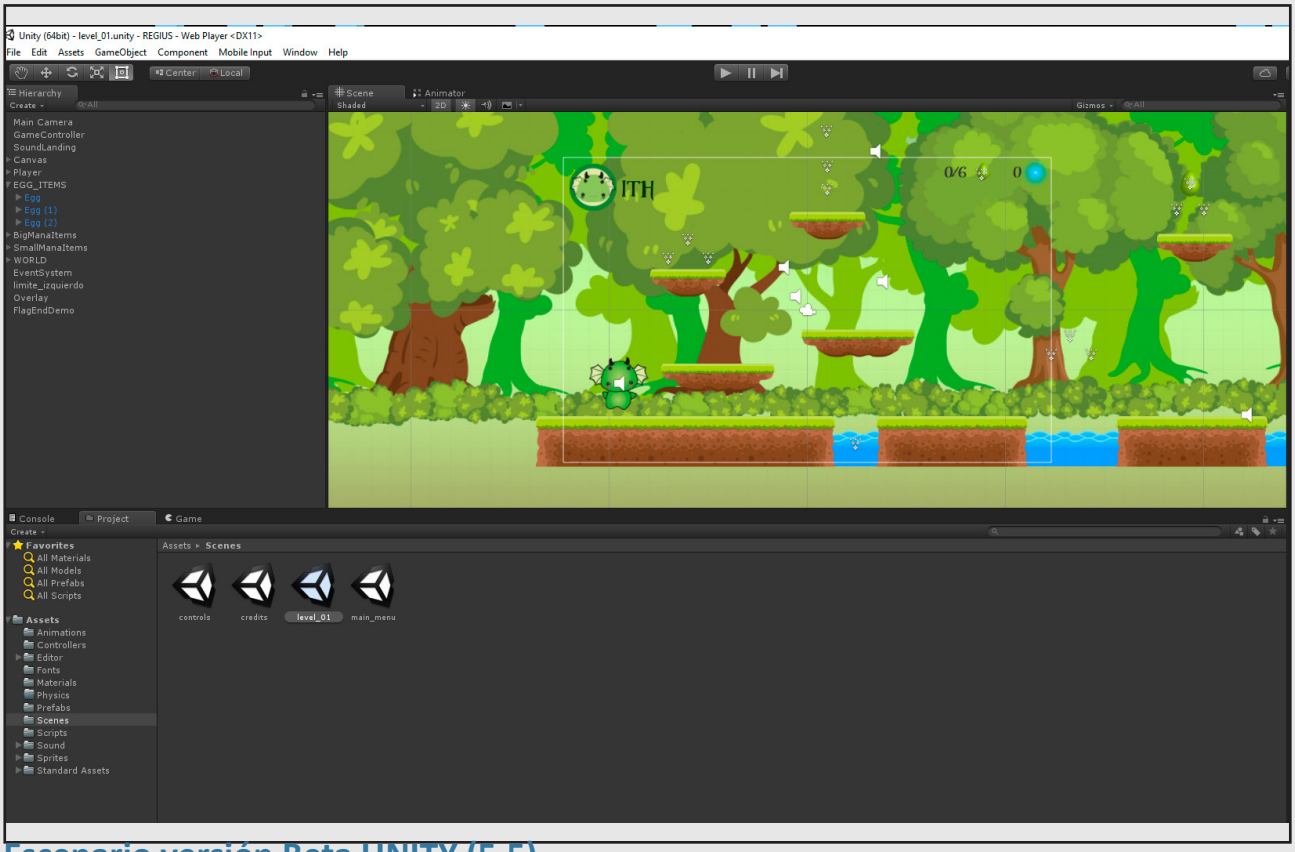

ITH
55


ITH
61

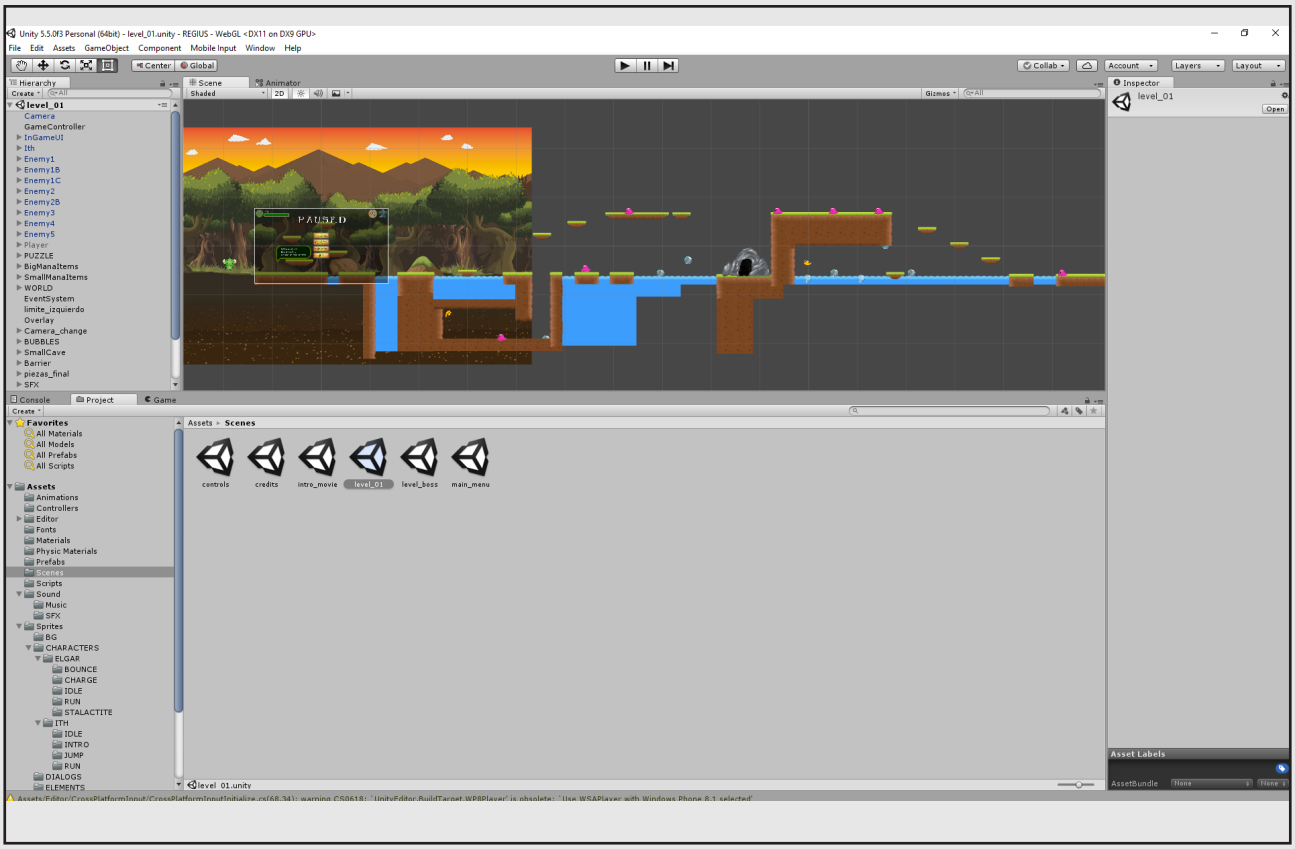

ITH
ELGAR
74


ITH
ELGAR
0

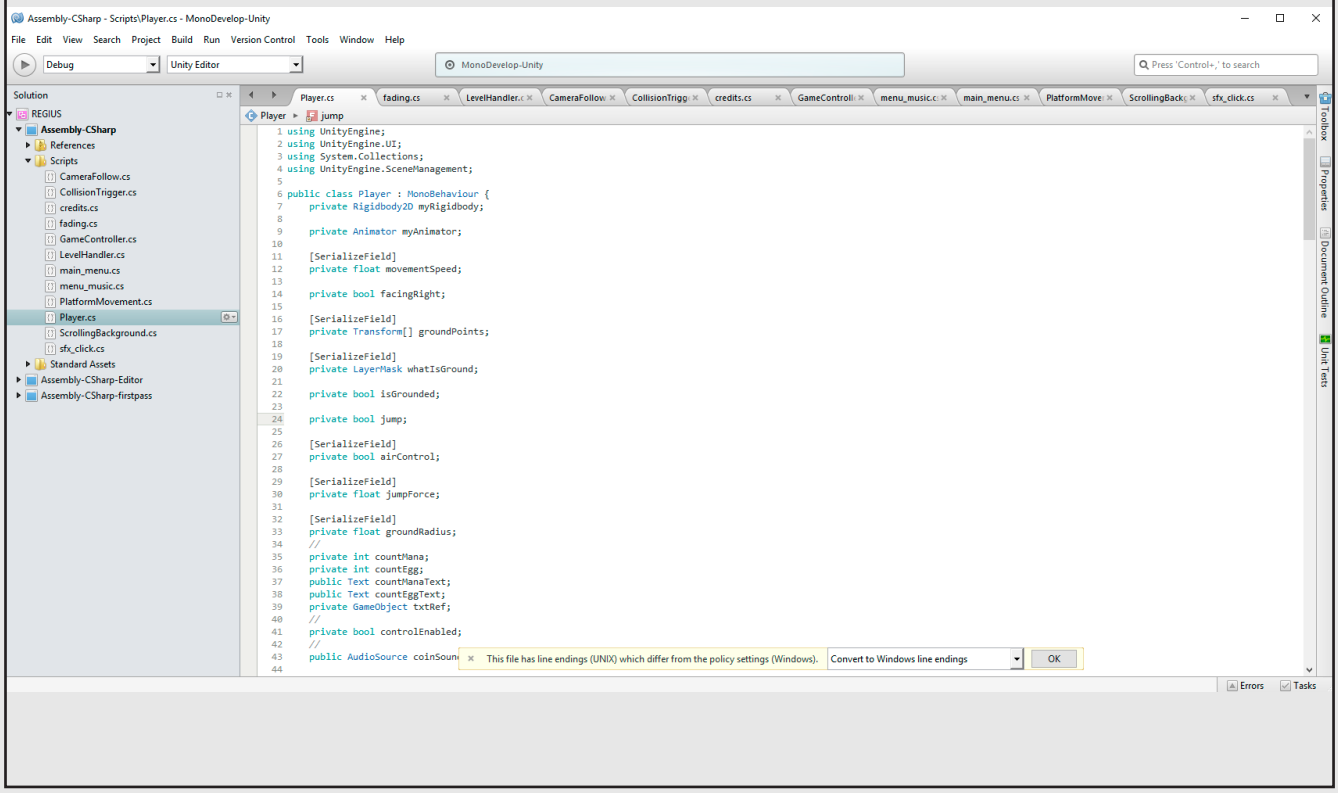
Escenario versión alpha en UNITY (5.3)



Escenario versión Beta UNITY (5.5)



Entorno programación MonoDevelop-Unity



INSTALACIÓN

REQUISITOS DEL SISTEMA

SO — Windows XP SP2+, Mac OS X 10.8+, Ubuntu 12.04+
Tarjeta gráfica — DX9 (modelo de shader 3.0) o DX11 con capacidades de funciones de nivel 9.3
CPU — Compatible con el conjunto de instrucciones SSE2
HDD — 600 Mb libres de espacio en disco

INSTALACIÓN

Descomprimir el archivo .rar o .zip (según versión descargada) en una carpeta del disco duro.
Ejecutar la el archivo Regius.exe (Windows)

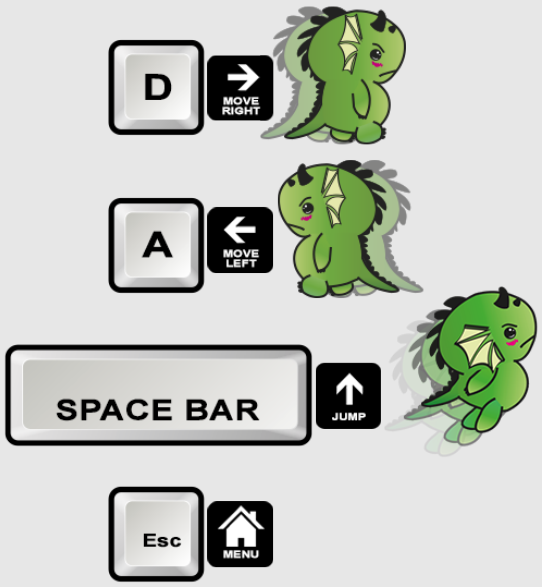
CONTROLES

MENÚ PRINCIPAL Y MENÚ DEL JUEGO

Utiliza el cursor del ratón para moverte por el menú principal del juego. Selecciona PLAY para iniciar el juego, controles para ver las teclas de control del personaje, ver créditos o salir.

Pulsa la tecla ESC durante el juego para que aparezca un menú de opciones, selecciona una opción en el menú para: volver al juego, reiniciar partida, volver al menú o salir del juego.

CONTROL DEL PERSONAJE

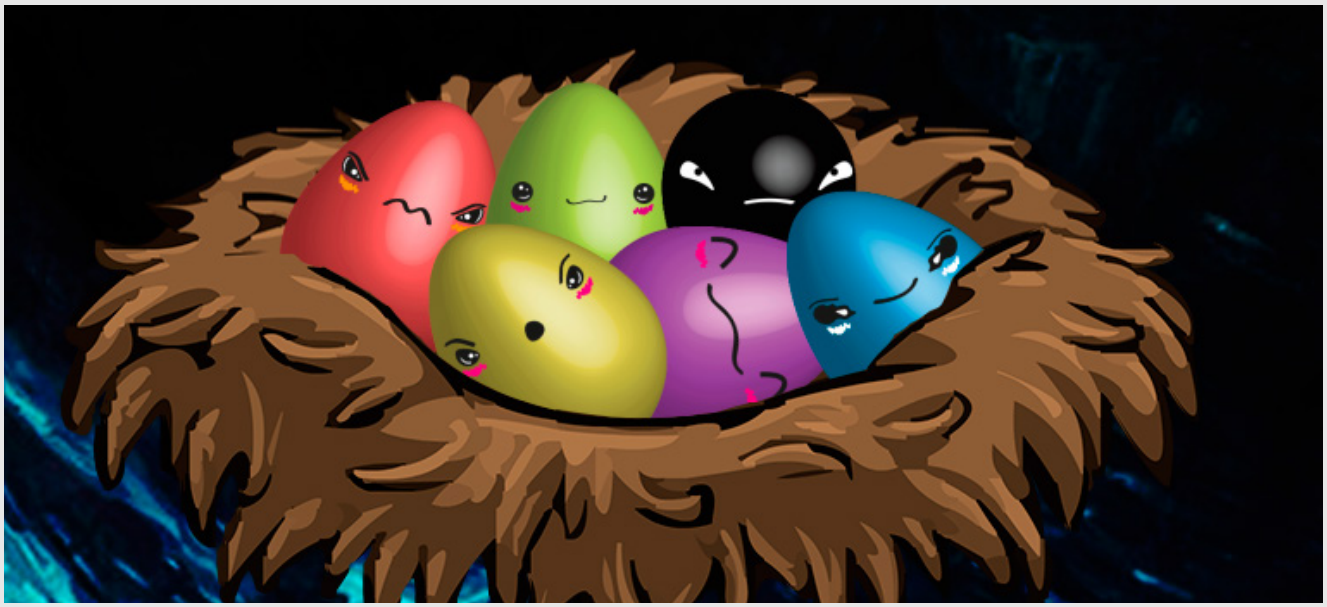


D - Manten pulsada la tecla para mover al personaje hacia la DERECHA.
A - Manten pulsada la tecla para mover al personaje hacia la IZQUIERDA.
ESPACIO - Pulsa la barra ESPACIO para saltar.
ESCAPE - Pulsa la tecla ESC para abrir el menú de opciones o para saltar la película introductoria.

Alternativamente puedes pulsar las flechas del teclado para mover el personaje.

OBJETIVOS

ARGUMENTO



ITH es un bebe dragón que debe buscar a sus hermanos que han sido secuestrados por un ser desconocido. En la búsqueda deberá enfrentarse a ellos para que recuperen la memoria. Cada dragón tiene una personalidad y habilidades diferentes que combinadas permitirán avanzar en esta aventura para rescatar al resto del nido y averiguar la identidad de su captor.

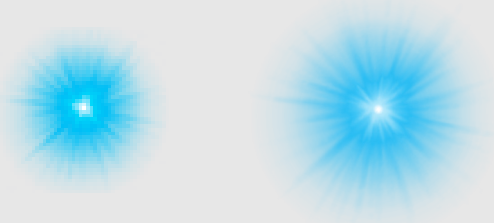
LOS PROTAGONISTAS

OBJETOS

PIEZAS DE PUZZLE

Consigue todas las piezas de puzzle para progresar y acceder a nuevas zonas, se encuentran escondidas por el mapa.

ESTRELLAS DE MANA



Consigue tantas como puedas para volverte mas ágil, puedes conseguir acumular hasta 100 para mejorar hasta un 40% tu velocidad de movimiento. Te serán de ayuda para derrotar al jefe final.
Las estrellas pequeñas aportan 1 unidad a tu contador de estrellas.
Las estrellas grandes aportan 5 unidades a tu contador de estrellas.

BURBUJAS

Salta sobre ellas para impulsarte.

ENEMIGOS

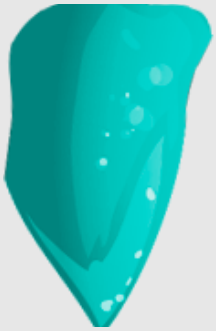
BABOSAS

Si te tocan perderás 10 puntos de vida. Evítalas o salta sobre ellas para eliminarlas.

AGUA

Ten cuidado y no caigas al agua, ITH no sabe nadar! Si caes perderás toda la vida y tendrás que volver a comenzar.

ESTALACTITAS



Caen de las cuevas y si te impactan te harán perder una buena cantidad de vida.

Imagen de la marca

RGB (224,90,90)
CMYK (0,76,55,0)
#E05A5A

RGB (199,226,89)
CMYK (35,0,80,0)
#C7E259

RGB (221,211,91)
CMYK (16,9,78,0)
#DDD35B

RGB (0,0,0)
CMYK (91,79,62,97)
#000000

RGB (179,91,221)
CMYK (45,68,0,0)
#B35BDD

RGB (90,163,224)
CMYK (75,16,0,0)
#5AA3E0

RGB (231,230,230)
CMYK (11,10,10,0)
#E7E6E6

RGB (255,255,255)
CMYK (0,0,0,0)
#FFFFFF

Tipografía del logo: EG Dragon Caps (minúscula)

LOREM IPSUM DOLOR SIT AMET, CONSECTETUR ADIPISCING ELIT. DUNC EGET PRETIUM TELLUS, A ULLAMCORPER EST. FUSCE VITAE EST UT NUNC PELLENTESQUE VARIUS. MAECENAS EX ORCI, PORTA NEC ULTRICES ID, RHONCUS LACINIA TURPIS.

Aunque el objetivo de la creación de este juego no es obtener ingresos económicos si podemos proyectar un escenario a futuro para analizar la viabilidad económica del mismo.

Los modelos de negocio más utilizados en la industria del videojuego son los siguientes:

Pay to play
El usuario tiene que pagar para poder instalar y jugar con la aplicación. Una vez realizado este pago el usuario es libre de jugar el tiempo que desee.

Gratuita con publicidad
Permite la descarga gratuita y los ingresos vienen por banners de publicidad de terceros. Esta publicidad puede estar integrada en el juego, aparecer durante el juego en determinados momentos o alcanzar un acuerdo con alguna marca y publicitarla dentro de forma subliminal.
Es un modelo que requiere una masa elevada de jugadores para obtener rendimientos económicos atractivos.

Freemium
Permite jugar de forma gratuita pero con limitaciones, ya sea por tiempo, funcionalidad, capacidad, uso, etc. Si el usuario desea saltarse estas restricciones debe pagar por la aplicación.

Micropagos
Permite la descarga gratuita pero algunos elementos del contenido son de pago. El objetivo de este modelo es atraer al jugador inicialmente y que una vez este dentro pueda comprar elementos no indispensables en el juego. Es un modelo que funciona muy bien para usuario/desarrollador ya que permite ajustarse a la capacidad económica de cada usuario.

Subscripción
El usuario debe realizar un pago periódico (mensual, anual...) para poder jugar. Suele ser un modelo utilizado en aplicaciones que requieren un mantenimiento y actualización de contenidos y que generan un gasto de servidores importante.

En nuestro caso, dado que hablamos de un juego indie de plataformas enfocado a un sector casual, los modelos que mejor se ajustarían a nuestro juego serían el **pay to play** o el de **micropagos**.

GLOSARIO

API
Interface de programación de aplicaciones (Application Programming Interface) es un conjunto de funciones y procedimientos recogidos en una biblioteca informática con la finalidad de ser utilizada por otro programa.

Bug
Nombre que se atribuye a un fallo, error en la aplicación o funcionamiento inesperado

C#
Lenguaje de programación desarrollado por Microsoft que forma parte del .Net Framework. Es un lenguaje multi-paradigma, orientado a objetos, funcional, genérico, imperativo y declarativo.

Detección de colisiones
Hace referencia a la resolución del problema computacional de detectar la intersección entre 2 objetos en un programa o juego.

Framework
Conjunto de estándares que forman una infraestructura para el desarrollo de software.

GUI
La interfaz gráfica de usuario (Graphic User Interface) es una interfaz que utiliza elementos gráficos que permite un control intuitivo del usuario a través de la interacción con esta.

Script
Archivo de texto escrito en un lenguaje de programación que contiene los comandos para realizar unos procedimientos como dotar de interactividad al juego.

Sprite
Imagen de dos dimensiones que se integra en una escena más grande dentro de un juego o aplicación.

Sprite sheet
Fichero compuesto por varios sprites que se organizan en un único fichero para aligerar la carga del procesador gráfico que trabaja mejor cargando una única imagen separándola en memoria que cargando cada una por separado.

BIBLIOGRAFÍA

Robert Nystrom (2014) Game Programming Patterns.
http://gameprogrammingpatterns.com/

Greg Lukosek (2016) Learning C# by Developing Games with Unity 5.x Second Edition
https://www.amazon.es/Learning-Developing-Games-Unity-Second/dp/1785287591/ref=sr_1_1?s=foreign-books&ie=UTF8&qid=1476874452&sr=1-1&keywords=learning+-c%23+by

Claudio Scoalstici (2015) Unity 2D Game Development Cookbook
https://www.amazon.es/Unity-2D-Game-Development-Cookbook/dp/1783553596/ref=sr_1_1?s=foreign-books&ie=UTF8&qid=1476874587&sr=1-1&keywords=unity+2d+-game+development+cookbook

Maciej Szcześnik (2016) Unity 5.x Animation Cookbook
https://www.amazon.es/Unity-Animation-Cookbook-Maciej-Szcze/dp/1785883917/ref=sr_1_1?s=foreign-books&ie=UTF8&qid=1476874655&sr=1-1&keywords=Unity+5.x+Animation

CONCLUSIÓN FINAL

La realización de este proyecto ha supuesto un enorme esfuerzo de aprendizaje tanto del motor UNITY, lenguaje C# y muchos de los procesos artísticos de diseño y animación de personajes. La experiencia es muy, muy positiva y pese al desgaste sufrido estoy muy satisfecho con el resultado final. Pese a todo si considero que el espacio de tiempo de ejecución es muy estrecho para conseguir unos resultados dignos.

Y es que la tarea de crear un videojuego es muy costosa ya que requiere del uso de muchas disciplinas. En el mundo laboral normalmente hay especialistas que ejecutan cada una de ellas, por lo que para una sola persona la realización de todas ellas requiere mucha polivalencia y esfuerzo.

Pese a todo, creo que fue una buena decisión “lanzarme a la aventura” de aprender a utilizar UNITY ya que a nivel profesional ahora me siento preparado para afrontar posibles proyectos realizados con ella. Lo cierto es que llevaba bastante tiempo queriendo aprender a utilizarla pero nunca encontraba el momento. A veces es necesario hacer uno de estos gestos, salir de la zona de confort, saltar a terreno desconocido y afrontar la decisión tomada.

He pasado por muchos momentos en este proyecto, muchos altibajos, ha sido una carrera de obstáculos que he ido superando paso a paso... me he caído algunas veces pero he continuado y por fin veo la meta final.

La verdad es que en la última etapa me he divertido mucho y quizás me he salido un poco del guion ya que se me ocurrían ideas y decidía implementarlas.
Espero que en un futuro no muy lejano la conclusión de este juego sea una realidad.

Y con este proyecto también acaba una larga travesía, una etapa. Ya voy pensando en la siguiente.

¡Gracias!

DOWNLOAD LINK

http://goo.gl/SdsgY2