Evolving

Jordi Farré

Resumen— En la sociedad actual se ha podido apreciar, en el transcurso de los últimos años, una situación de crecimiento y expansión del sector tecnologíco y la economía y el mercado del mismo, pudiendo presenciar el nacimiento y, en algunos casos, éxito, de nuevas empresas -o *StartUps*-. EsteTrabajo de Fin de Grado (de ahora en adelante, TFG) consistirá en proponer, estudiar y desarrollar una idea con el potencial de convertirse el foco central de una nueva *StartUP*. El proyecto que se ha llevado acabo en la realización de este TFG ha sido la creación de un juego *indie* de plataformas 2D, lo que ha incluido todas las estapas del desarrollo: comparación entre distintas ideas, selección del proyecto y estudio de mercado, decisión de las herramientas más adecuadas para el trabajo a realizar, documentación y captación de requisitos, diseño del modelo de negocio, creación de de maquetas y *storyboards* y, finalmente, la implementación del juego.

Palabras clave—videojuego, plataformas, 2D, unity, desarrollo, programación, *prefab,* componente, *StartUp*, modelo de negocio, escenas, nivel

Abstract— In the course of recent years, there has been a situation of growth and expansion in the technology sector and its economy and market, being able to witness the birth and, in some cases, success, of new companies -or StartUps-. This Final Degree Project (hereinafter, TFG) will consist of proposing, studying and developing an idea with the potential to become the central focus of a new StartUP. The project slected for the realization of this TFG has been the creation of an indie 2D platform game, which has included all stages of development: comparison between different ideas, selection of the project and market study, decision of the most appropriate tools for the work to be carried out, documentation and capture of requirements, design of the business model, creation of mockups and storyboards and, finally, the implementation of the game.

Index Terms—videogame, platforms, 2D, unity, development, prefab, component, StartUp, scenes, bussines model, level

1 Introducción

ESTE proyecto consiste en proponer y desarrollar una idea alrededor de la cual se pueda construir una Start-Up. Inicialmente, había dos propuestas distintas de proyectos consistentes en:

- Una app con un catálogo de series y películas que te indique en qué plataformas de streaming pueden encontrarse cada una de ellas, y que te redireccione a éstas. Además, se podrían hacer listas de las series que te gustan, así como comentarlas y/o escribir críticas y opiniones.

- Hacer un videojuego de plataformas en 2D, cuya característica principal sea la evolución y la adaptación: los enemigos cada vez se van haciendo más fuertes y mejoran sus estrategias para defenderse contra el jugador y derrotarle; eso incluye hacerse más resistentes a los ataques más utilizados, cambiar sus patrones de ataque en función de combates anteriores y de su estilo de juego, etc.

Después de hacer un estudio de mercado, tecnológico y una comparativa de ambas ideas, finalmente se ha decidido realizar el desarrollo del videojuego.

1.1. Viabilidad: estudio de mercado y aspectos técnicos

En el aspecto técnico, esta idea no requiere de mucha variedad de técnicas, puesto que principalmente se ha utilizado el motor gráfico Unity, para el desarrollo del videojuego y alguna otra herramienta para la interfaz de la aplicación.

Algunas de las complejidades que ha entrañado este proyecto a nivel personal han sido la falta de experiencia desarrollando videojuegos de estas características y no haber trabajado nunca en aspectos como las inteligencias artificiales de los NPC o el diseño de niveles; que entraña cierto nivel de dificultad si no se tienen los conocimientos o la práctica suficiente.

Otro de los problemas de esta idea, es el diseño y el aspecto gráfico de los personajes y de los entornos. Esto, para la etapa del desarrollo en la que se encuentra *Evolving*, se ha solucionado mediante la utilización de assets gratuitos de la store de Unity, y si el proyecto finalmente continuase, se podría contar con un diseñador que lo hiciese de manera profesional.

El aspecto positivo es que, aunque se ha pasado por un proceso de aprendizaje para utilizar Unity, el tiempo necesitado para esto se ha visto reducido, ya que se contaba con experiencia en la programación con C#, que es el lenguaje utilizado en este motor gráfico.

En cuanto al mercado, el sector de los videojuegos es

muy vasto, con muchísimos competidores, pero a su vez, con una inmensa cantidad de usuarios potenciales.

Y después de investigar, aunque se observaron muchos juegos con conceptos e ideas parecidas, no hay ninguno exactamente igual, eso si se compara con la idea del producto final, no del prototipo.

1.2. Descripción y funcionamiento del juego

Evolving se trata de un juego en 2D del genero de las plataformas. El objetivo principal es superar una serie de niveles, o escenarios, que contarán con obstáculos que el jugador tendrá que evitar y enemigos a los que combatir, los cuales le pueden herir.

El personaje principal se trata de un caballero que cuenta con una espada y es capaz de realizar movimientos de desplazamiento lateral, salto y un ataque básico.

Éste cuenta con una barra de salud, la cual indica el estado del mismo; si es dañado, la vida disminuirá y al llegar a 0, perderá una de sus vidas. Cuando no le quede ninguna, se mostrará la pantalla de *Game Over* (fin de la partida), y se nos dará la opción de volver al menú principal.

Cada vez que se pierda una vida, se volverá a cargar el escenario; en caso de que el jugador haya pasado por algún *Checkpoint*, empezará desde ese punto, y en caso de no ser así comenzará desde el principio del nivel.

Los enemigos, de igual manera, también pueden ser eliminados, mediante los ataques de nuestro personaje, si su salud llega a 0. Los distintos tipos de oponentes tienen diferentes valores de salud.

Además, estos también hacen más o menos daño y se comportan y se mueven con distintos patrones.

Si conseguimos llegar al final del nivel, aparece una pantalla que nos avisa de la victoria y podemos volver al menú principal.

En el menú, tenemos varias opciones:

- Selección de niveles, que nos deja escoger entre los escenarios implementados, con distintas dificultades y enemigos
- Menú de opciones, desde el cual podemos cambiar aspectos como la resolución, los gráficos o el volumen.
- Salir, que nos permite finalizar el juego.

2. Estado del arte

El mundo de los videojuegos es muy amplio, con una gran cantidad de usuarios, productos y plataformas en las que jugarlos.

Una de las decisiones más importantes que hay que tomar a la hora de desarrollar un juego es, precisamente, para qué plataforma quiere implementarse. Esto puede determinar una serie de factores, como por ejemplo cómo será el desarrollo, qué herramienta se utilizará, cómo se diseñará la interfaz o los controles, entre otras cosas.

Entre las múltiples plataformas para las que se puede desarrollar el juego, las más destacadas hoy en día son: PS4, PS5, Nintendo Switch, Xbox Series X|S, PC o móvil.

El resultado ideal, y al que aspira Evolving en caso de convertirse en el buque insignia para una nueva Start-Up, sería poder ser jugado en cualquier plataforma, es decir, ser un juego multiplataforma. Aunque esto tiene un mayor grado de complejidad y requiere de más tiempo, herramientas como Unity -el motor utilizado para desarrollar el proyecto-, permiten el desarrollo de juegos multiplataforma y tienen una serie de herramientas para poder hacerlo. Aún así, en esta primera etapa del proyecto, se presentará una versión funcional solo para una plataforma -PC-.

El motivo por el cual se ha hecho esta elección es que PC es uno de los dispositivos más populares entre los consumidores de videojuegos, y gracias a la existencia de plataformas como Steam, Evolving, como modelo de negocio, sería mucho más viable. Por otro lado, no se dispone de ninguna de las consolas más populares actualmente en el mercado, por tanto, sería imposible testear el juego conforme se avanza en sus etapas de desarrollo.

3. Objetivos

Debido a la naturaleza de este TFG, siendo un producto que pretende ser el puntal para la creación de una *StartUp*, nos encontramos ante un proyecto extenso y ambicioso, que, además, será desarrollado por una única persona. Debido a esto, se ha diferenciado el resultado final de los objetivos que pretenden ser alcanzados durante la duración de este TFG, que son los siguientes:

- Interfaz básica del juego
- Menú principal
- Menú interno del juego
- Un único personaje con sus movimientos y habilidades (básicas)
- Distintos tipos de enemigos (2 o 3), cada uno con diferentes patrones de movimiento y tipos de ataque
- 1 o 2 niveles distintos
- Una primera implementación básica de la mecánica principal del juego, la evolución: los enemigos se hacen más resistentes a los ataques que utilices, o varían ligeramente su patrón de ataque o directamente son sustituidos por una versión avanzada de ellos mismos, etc.

Estos serían los requisitos principales del juego, y los objetivos que se han marcado para este trabajo.

4. Desarrollo

El primer punto a trabajar en el desarrollo de este proyecto ha sido, tal y como ya se ha comentado en

apartados anteriores, la selección de una idea y un estudio de viabilidad de la misma. Después de esto, se ha hecho una búsqueda y comparativa de las distintas herramientas que se utilizarán para la realización de este TFG.

Con las herramientas ya seleccionadas y antes de proceder a la implementación de *Evolving*, se documentó el proyecto y se hizo una captación de los requisitos. También se diseñó una maqueta y un storyboard, que mostraban cual sería el aspecto y el funcionamiento del juego.

Simultáneamente, se estaba aprendiendo a manejar la herramienta Unity y se realizaron tutoriales para familiarizarse con ella.

Finalmente, el siguiente paso fue comenzar con el desarrollo del juego, concretamente con la implementación del personaje principal y sus respectivos movimientos. Después de esto, se añadieron los primeros escenarios y enemigos, el sistema de combate y de salud, los menús, etc.

4.1. Herramientas seleccionadas

Antes de comenzar con el desarrollo del juego, se procedió a hacer un estudio de las herramientas y tecnologías principales que se utilizarían.

Motor gráfico

En el desarrollo de un videojuego, una de las piezas centrales es el motor gráfico. Éste es un entorno que suele englobar una serie de rutinas de programación y herramientas que permiten el diseño, creación y funcionamiento de un videojuego.

Por tanto, elegir qué motor gráfico se utilizaría para el desarrollo de Evolving fue una de las decisiones más importantes que se ha tenido que tomar antes de empezar a desarrollar el proyecto.

Por ello, se buscó información y se consultó cuales son actualmente las mejores herramientas. De entre éstas, se escogieron las cinco mejores y más populares. La comparación se puede encontrar en el apartado 8.1 de los anexos.

Tal y como se puede apreciar en la tabla, y después de consultarlo con profesionales del sector, se concluyó que los dos mejores candidatos para el desarrollo de *Evolving* eran Unity y Unreal Engine. Ambos son motores gráficos muy potentes, que suelen ser la principal opción de la mayoría de estudios y desarrolladores que no disponen de una herramienta propia. Los dos son gratuitos, ofrecen una gran cantidad de herramientas y permiten el desarrollo de videojuegos multiplataforma.

Para poder decidir entre uno y otro, se valoraron las principales diferencias entre ambos. Unity, ofrece un entorno de trabajo más sencillo y entendible y, de los dos, es el que tiene una mejor curva de aprendizaje. Asimismo, el principal lenguaje de programación para desarrollar en Unity es C#, frente a C++, que es el que utiliza Unreal. C# es mucho más sencillo, además de disponer de muchas bibliotecas y librerías que pueden facilitar la tarea de desarrollo.

Por otro lado, aunque tal y como se ha podido comprobar, Unreal es un entorno más complejo y trabaja con un lenguaje de bajo nivel como es C++, una de las principales ventajas de esta herramienta es su potencia. Este motor gráfico permite el desarrollo de juegos muy avanzados (triple A), con excelentes gráficos y que emplean unas mecánicas y unas físicas muy complejas. Además, Unreal también es de código libre, esto significa que aquellos desarrolladores que así lo deseen, pueden modificar el código y adaptar la herramienta a sus necesidades.

Teniendo en cuenta lo comentado anteriormente, se decidió elegir Unity, ya que debido a las características de *Evolving*, no se necesitan las grandes ventajas que ofrece Unreal, pues el juego es más sencillo, con gráficos más básicos y mecánicas no tan complejas, algo así como un juego indie. Debido a esto, se prefirió elegir la opción que ofrece un entorno de trabajo más simple, además, como aliciente, ya se cuenta con experiencia a la hora de desarrollar en C#, lo cual puede agilizar el trabajo.

Diseño y maquetación

En la fase previa de este proyecto, antes de que se comenzase con el desarollo, se decidió hacer un diseño inicial de como se verían las diferentes pantallas del videojuego, es decir, un prototipaje digital e interactivo. Este prototipo es el Mínimo Demostrador Viable de este proyecto, que es la expresión mínima que puede enseñarse de un producto.

Para esto, se ha tenido que buscar y decidir cuáles son las herramientas más adecuadas para ello. Hay un sinfín de maneras de realizar esta tarea, pero algunas de las mejores son mediante: Marvel Apps, Balsamiqq, Justinmind, Wix, InVision, Adobe XD. Aunque inicialmente la candidata más firme era Marvel Apps, puesto que se tenía experiencia en la utilización de esta herramienta, finalmente fue descartada, ya que su uso principal es el prototipado de Apps y Webs, y no de videojuegos -aunque podría hacerse-.

Con esto último en mente, las candidatas más firmes fueron InVision y Adobe XD, puesto que dan más facilidades para el maquetado de videojuegos. De entre estas dos, finalmente, se optó por Adobe XD, ya que su versión gratuita está menos limitada que la de InVision.

Teniendo ya la herramienta seleccionada, se procedió a diseñar un prototipo interactivo, el cual permite visualizar como sería el aspecto aproximado del juego y la interacción entre las distintas pantallas, las cuales pueden verse en el apartado 8.2 anexos.

4.2. Captación de requisitos y documentación

La búsqueda de información y la documentación son aspectos clave para el desarrollo de un proyecto, sobre todo en su inicio. La creación de documentos que recojan las características, las necesidades de los usuarios y los stakeholders, así como los requisitos, es de suma importancia, y más si se trata del desarrollo de un producto software, que es menos tangible y más difícil de visualizar para todos los implicados.

Es por eso que, además de todo lo realizado con anterioridad en este informe, se ha decido crear dos documentos que plasmen los aspectos mencionados anteriormente, entre otros.

Estos informes son: el Documento de Visión y el de Especificación de Requisitos del Software (SRS).

En el primero, se puede encontrar una explicación de la oportunidad de negocio y la posición del producto en el mercado, una definición de stakeholder y de usuario del sistema, así como sus perfiles y necesidades. También consta de una visión general del sistema, que incluye un árbol de objetivos y un resumen de las características del juego, así como de una descripción general del sistema, que contiene, además, un diagrama de casos de uso.

En el segundo, se haya una descripción general del proyecto, los requisitos específicos y los distintos casos de uso del sistema, así como una modelización del sistema en base a estos. Por último, información de apoyo como imágenes del prototipo de la aplicación.

4.3. Storyboard

A la hora de desarrollar un videojuego, es muy importante hacer antes un diseño conceptual; un dibujo, sketch o esbozo de como se vería el juego que está en proceso de creación.

Esto sería similar al prototipo interactivo mostrado en uno de los apartados anteriores, con la diferencia de que en este diseño se muestra, paso a paso, el aspecto y la evolución del juego o de un nivel de éste.

Este dibujo, llamado storyboard, es en realidad una sucesión de dibujos estáticos que nos muestran el avance del personaje principal de *Evolving* a través del juego, concretamente, del primer nivel.

En este storyboard, se puede ver como el primer nivel es en realidad un tutorial diseñado para mostrar al jugador las mecánicas principales del juego y los movimientos del personaje. Por tanto, tal y como se puede apreciar en el diseño conceptual, este tutorial es muy sencillo y con pocas pantallas.

Aunque solo se diseñó el primer nivel para la fase inicial de este proyecto, finalmente se han desarrollado tres niveles con distintos enemigos y pruebas.

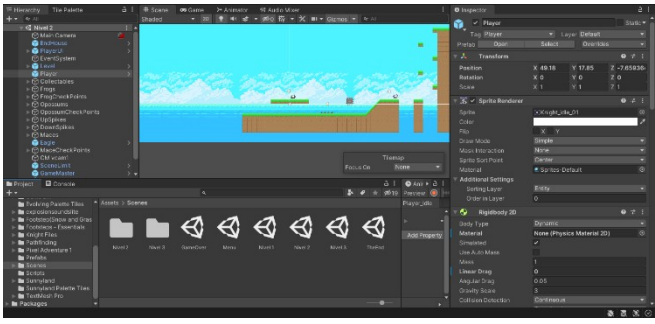


Ilustración 1: Editor de Unity y sus componentes

4.4. Metodología y funcionamiento del juego

En esta sección, se explicará la metodología que se ha seguido para la implementación de *Evolving*, como funciona Unity y los distintos elementos del juego.

Unity

Tal y como ya se ha explicado en los anteriores apartados de este informe, la herramienta utilizada para el desarrollo de *Evolving* es Unity, uno de los mejores y más populares motores gráficos existentes, y que además su uso es libre y gratuito.

La manera de construir un juego en Unity es mediante *Scenes* -escenas-, estas escenas es lo que el usuario vería en pantalla cuando juega, contienen los distintos elementos del juego, y normalmente cada una de éstas correspondería a un escenario o nivel.

Dentro de estas escenas, se agregan los *GameObjects* (GO), que son las principales instancias dentro de cada escena, y representan todos los posibles elementos del juego: personajes, enemigos, obstáculos, plataformas, los bloques con lo que se construye el escenario, etc.; cada elemento que quiera ser añadido, corresponderá con un *GameObject*.

Además, estas instancias constarán a su vez de componentes que pueden ser agregados o eliminados dependiendo de lo que queramos implementar en nuestro juego. Algunos de los componentes utilizados en la construcción de *Evolving* son:

- Transform: Este es un componente básico que viene por definición en todos los GO, y que indica el tamaño y la posición del mismo dentro de la escena.

- Sprite Renderer: Es el aspecto visual que tendrá nuestra instancia, ya sea el dibujo del personaje, los enemigos o los bloques con los que se construyen las plataformas y superficies.

- Rigidboy 2D: Es lo que confiere al GO de físicas propias como, por ejemplo: que pueda moverse, que se vea afectado por la gravedad, etc.

- Boxcollider 2D: Es lo que dota al elemento de la capacidad de interactuar con otros elementos, o lo que es lo mismo, permite detectar si colisiona con otro objecto.

- Animator: Este componente permite crear animaciones y transición entre ellas, para que los personajes o enemigos tengan distinto aspecto según si saltan, se mueven, atacan, etc.

- Script: En este componente es donde se encuentra, principalmente, la programación y el código desarrollado para este proyecto. Mediante estos scripts, es como se logra que el resto de componentes funcionen y que los GO puedan interactuar entre ellos, con la escena y con el resto del juego.

En la siguiente figura, se puede ver un ejemplo de como se trabaja en Unity: en el centro se encuentra la escena en la que se está trabajando y en el lateral izquierdo están los GO que la componen, y por último, a la derecha está el Inspector, que nos muestra la información de cada GO y de los componentes que contiene. Además, en la parte inferior pueden ubicarse otras ventanas, como la que contiene la estructura de carpetas del proyecto.

Implementación

Ahora que ya se ha comentado el funcionamiento de Unity, se procederá a explicar la implementación de *Evolving* utilizando éste.

A continuación, se muestra la composición de *GameObjects* que tiene una de las escenas del juego, concretamente, el primer nivel que se desarrolló.

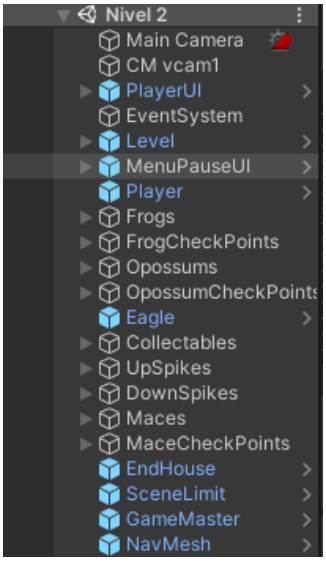


Ilustración 2: GameObjects en una escena de Evolving

MainCamera

Es uno de los objetos más básicos y principales, y viene por defecto a la hora de crear una escena. Cumple la función de cámara, ya que se desplaza por la escena y aquello que enfoca es lo que el jugador verá por la pantalla.

Como *Evolving* es un juego 2D, la cámara quedará fija en una de las coordenadas, de manera que todo el rato se enfocará a un plano y a los elementos que contiene.

CM vcam1

Este GO es una instancia del objecto de Unity *Cinemachine*, y, en esencia, se trata de una cámara virtual que realiza una serie de configuraciones que permiten mostrar mejor la escena.

MainCamara inicialmente está estática, lo cual es un problema si queremos mostrar como el personaje principal se mueve a lo largo de un escenario grande; una de las funciones que es capaz de realizar esta instancia de *Cinemachine* es hacer un seguimiento a un GO concreto, que en este caso será nuestro personaje.

Además, se pueden modificar otros valores como, por ejemplo, la velocidad y el movimiento de la cámara, la distancia, la perspectiva, la posición relativa del objeto que se está siguiendo, etc.

Ilustración 3: Componentes del GO Foreground

Level

En Unity, para poder crear el mundo o los escenarios por los cuales se moverá el jugador, se necesitan una serie de elementos gráficos llamados *Tiles*. Estos objetos son bloques que permiten construir cualquier componente del escenario de nuestro juego, desde el fondo (*Background*) hasta las plataformas, el suelo y los obstáculos.

Para poder hacer esto, se necesita crear un GO llamado *Tilemap*, que es una cuadrícula que permite colocar de forma sencilla los *Tiles* y dotarles de distintas características mediante componentes de Unity.

Nuestro objeto Level, consta de dos *Tilemaps* distintos: uno que se encarga del fondo del escenario (*Background*) y otro en el que se colocan los propios elementos del mismo (*Foreground*).

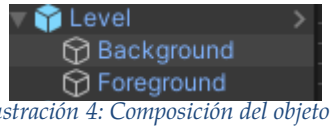


Ilustración 4: Composición del objeto Level

Mientras que *Background* solo consiste en imágenes, *Foreground* tiene más componentes que permiten al jugador interactuar con el escenario.

Tilemap Collider y *Composite Collider* sirven para que nuestro personaje principal pueda colisionar y "tocar" los

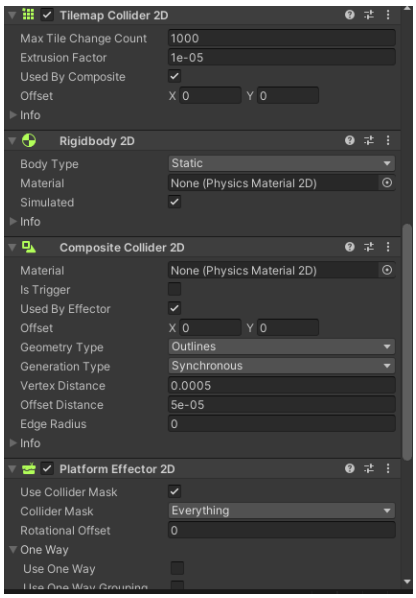
bloques que conforman nuestro *Tilemap*, es decir, el suelo, las paredes, las plataformas, etc.

Rigidbody es lo que permite que haya físicas como la gravedad y que los bloques se mantengan en su posición en vez de que entren en contacto con otros objetos, como por ejemplo el jugador.

Por último, *Platform Effector*, permite que la fricción con los laterales de las plataformas sea más fluida.

PlayerUI

En un juego, las interfaces de usuario, o UI (del inglés, *User Interface*), son fundamentales, ya que muestran



información necesaria para el usuario. *PlayerUI* se encarga de mostrar cuanta salud tiene el jugador, la cantidad de vidas que le quedan y los objetos coleccionables que ha recogido hasta ahora.

Esta interfaz se ha implementado utilizando un *GameObject* llamado *Canvas*, que está pensado para interfaces y diseños. Se trata de un "lienzo" que permite dibujar y colocar elementos.

La información que muestra *PlayerUI* por pantalla, la envían los scripts encargados del funcionamiento de nuestro personaje principal y de los coleccionables.

MenuPauseUI

Este objeto es el menú de pausa, y de la misma manera que *PlayerUI*, es una interfaz de usuario creada con el GO *Canvas* y con distintos elementos gráficos como botones y texto.

Además, uno de los componentes que tiene es un Script que controla su comportamiento.

Las funciones que realiza este menú son:

- Al pulsar la tecla Esc, el juego se pausa y se muestra el menú con todas sus opciones
- "Reanudar" permite salir del menú y volver al juego

```
private void HorizontalMovement()
{
    float horizontalDirection = Input.GetAxis("Horizontal");

    if (horizontalDirection < 0)
    {
        playerRigidBody.velocity = new Vector2(-playerSpeed, playerRigidBody.velocity.y);
        GetComponent<SpriteRenderer>().flipX = true;
    }
    else if (horizontalDirection > 0)
    {
        playerRigidBody.velocity = new Vector2(playerSpeed, playerRigidBody.velocity.y);
        GetComponent<SpriteRenderer>().flipX = false;
    }
}
```

- La opción "Menú principal" saca al jugador del escenario en el que esté jugando y lo lleva al Menú principal del juego.
- "Menu de opciones" muestra otro menú distinto, que permite configurar aspectos como el sonido, la resolución o la calidad gráfica.
- Finalmente, "Salir" termina la ejecución del juego.

Ilustración 5: Método para el movimiento horizontal del jugador

Ilustración 6: Menú de pausa de Evolving

Player

Este es uno de los *GameObject* principales del juego y se trata, básicamente, del objeto que representa al personaje principal.

Está compuesto por unos *sprites*, que son su diseño gráfico, y distintos componentes que le permiten realizar todas sus funcionalidades.

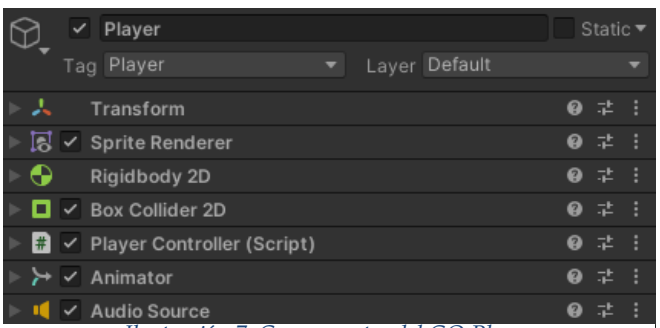


Ilustración 7: Componentes del GO Player

La función de *Rigidbody* y *Box Collider* ya se ha comentado en apartados anteriores, y, en resumen, es lo que confiere al personaje la capacidad de interactuar y colisionar con otros objetos y de tener físicas, como velocidad y gravedad.

Animator, es un componente que permite transicionar entre una animación y otra, es decir, es una maquina de estados que según ciertos parámetros que se le indican, muestra distintas animaciones: correr, saltar, atacar, etc.

Audio Source, permite gestionar los efectos de sonido emitidos por nuestro GO, como por ejemplo el de andar sobre la hierba o saltar.

Y por último, comentaremos algunas de las funciones del script *PlayerController*.

Esta función es la que controla el desplazamiento horizontal de *Player*. Lo que hace es aumentar la velocidad en la coordenada x en función de si se están pulsando las teclas que gestionan el eje horizontal, y, según la dirección del jugador, también rota el *Sprite*.

Este método llama a la función *Jump*(), cuando se está pulsado el botón asignado para esta acción y si *Player* está tocando el suelo. También deja hacer un doble salto, en caso de que el jugador esté en el aire y solo haya saltado una vez.

```
private void Jumping()
{
    if (Input.GetButtonDown("Jump"))
    {
        if (playerCollider.IsTouchingLayers(ground))
        {
            Jump(State.Jumping);
        }
        else if (state == State.Jumping || state == State.Falling)
        {
            Jump(State.DoubleJumping);
        }

    }
}
```

Ilustración 8: Funcionalidad de salto de Player

Esta función, una vez es pulsado el botón para atacar (la barra espaciadora), activa la animación de ataque y detecta si ha habido alguna colisión con algún objeto de tipo enemigo. Si es así, el enemigo recibirá daño.

Estás no son los únicos métodos que hay, y el script contiene otras implementaciones, como la colisión con objetos coleccionables u obstáculos.

```
private void Attacking()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        state = State.Attacking;
        playerAnimator.SetTrigger("Attack");

        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange, enemyLayers);

        foreach (Collider2D enemy in hitEnemies)
        {
            enemy.GetComponent<Enemy>().ReceivingDamage(attackDamage);
        }

        attackTime = Time.time + 1f / attackRate;
    }
}
```

Enemigos: Opossums, Frogs y Eagles

En esta implementación actual de *Evolving*, tenemos 3 tipos diferentes de enemigos: oposums (zarigüeyas), frogs (ranas) y eagle (águila).

Los enemigos tienen distintos tipos de patrones de movimiento, pero su principal función es infligirle daño al *Player* cuando éste entra en contacto con ellos.

Comparten algunos elementos comunes con el *Player*, como por ejemplo un *Rigidbody*, un *Collider*, un *Animator* y un *AudioSource.*

Además, cuentan con dos Scripts que implementan sus funciones básicas:

- Enemy.cs, que es compartido entre todos los tipos de enemigos

```
if (transform.position.x > leftCheckPoint.position.x)
{
    if (enemyCollider.IsTouchingLayers(ground))
    {
        enemyRigidBody.velocity = new Vector2(-jumpLength, jumpHeight);
        enemyAnimator.SetBool("IsJumping", true);
    }
}
else
{
    GetComponent<SpriteRenderer>().flipX = true;
    rightOriented = true;
}
```

- Los scripts propios (Opossums.cs, Frog.cs, Eagle.cs) que heredan de Enemy.cs e implementan funcionalidades propias para cada enemigo.

A continuación, se explican algunas de las funciones.

Ilustración 10: Fragmento de código para el salto de Frog

```
public void ReceivingDamage(int damageReceived)
{
    currentHealth -= damageReceived;
    if (currentHealth <= 0) Die();
}

1 referencia
private void Die()
{
    enemyAnimator.SetTrigger("Death");
    deathAudio.Play();
    enemyRigidBody.velocity = Vector2.zero;
}

0 referencias
private void Death()
{
    Destroy(this.gameObject);
}
```

Ilustración 11: Métodos de la clase común Enemy

Estos tres métodos, son compartidos entre todos los enemigos, y básicamente gestionan el daño recibido, así como la muerte del enemigo.

De ellas, comentaremos con un poco más de detalle la

función *Die*(), ya que las otras dos son bastante sencillas y más intuitivas.

Die() se encarga de activar la animación de la muerte del enemigo, además de reproducir un audio específico.

```
private void Move()
{
    if (!rightOriented)
    {
        if (transform.position.x > leftCheckPoint.position.x)
        {
            if(enemyCollider.IsTouchingLayers(ground))
            {
                enemyRigidBody.velocity = new Vector2(-opossumSpeed, enemyRigidBody.velocity.y);
            }
        }
        else
        {
            GetComponent<SpriteRenderer>().flipX = true;
            rightOriented = true;
        }
    }
    else
    {
        if (transform.position.x < rightCheckPoint.position.x)
        {
            if (enemyCollider.IsTouchingLayers(ground))
            {
                enemyRigidBody.velocity = new Vector2(opossumSpeed, enemyRigidBody.velocity.y);
            }
        }
        else
        {
            GetComponent<SpriteRenderer>().flipX = false;
            rightOriented = false;
```

Ilustración 12: Función para el movimiento de Opossum

Esta es la función principal del enemigo *Opossum*, y gestiona su movimiento. Este enemigo se mueve de un punto a otro del escenario, solo si se encuentra tocando el suelo. Para conseguir esto, miramos donde se encuentra y comparamos su posición relativa a estos dos puntos, y aumentamos su velocidad en una dirección u otra, además de rotar su Sprite.

De la misma manera, hacemos algo muy similar con *Frog*, ya que tiene un patrón de movimiento parecido, pero saltando. Por tanto, se compara su posición relativa y solo si está en el suelo, hacemos que salte.

Por último, *Eagle* tiene un patrón de movimiento bastante distinto a los otros dos tipos de enemigos, ya que se mantiene volando en el aire, y cuando Player entra dentro de su rango empieza a perseguirle para atacarle e infligirle daño.

Para poder hacer esto, tiene un componente llamado *NavMesh* que le permite navegar por el escenario evitando obstáculos y realizando un algoritmo de *Pathfinding* hacía el jugador.

El método que hace que esté quieto o persiga al usuario en función de la distancia, es el siguiente.

```
private void Update()
{
    if (Vector2.Distance(transform.position, target.position) < distance)
    {
        agent.SetDestination(target.position);

    }
    else
    {
        if (!(Vector2.Distance(transform.position, currentPosition) <= 0))
            transform.position = Vector2.MoveTowards(transform.position, currentPosition, agent.speed * Time.deltaTime);
    }
}
```

Ilustración 14: Funcionalidad del movimiento de Eagle

Compara la distancia entre ambos, y si es más pequeña, pone en marcha a Eagle, y si no le devuelve a su posición inicial.

Obstáculos: spikes y maces

Además de enemigos, el jugador también se encontrará a lo largo del escenario con distintos obstáculos que ha de superar y que le pueden eliminar si no lo hace.

Dos de estos obstáculos son los *Spikes* y *Maces.* Un elemento que comparten ambos es un *Collider*, componente que les permite detectar si hay una colisión con el *Player* y hacerle daño si es así. A continuación, el fragmento de código que lo permite.

```
if (collision.gameObject.tag == "Obstacle")
{
    state = State.Hurt;
    TakeDamage(obstacleDamage);

    if (collision.gameObject.transform.position.x > transform.position.x)
    {
        playerRigidBody.velocity = new Vector2(-playerHurtForce, playerRigidBody.velocity.y);
    }
    else
    {
        playerRigidBody.velocity = new Vector2(playerHurtForce, playerRigidBody.velocity.y);
    }

    if (collision.gameObject.transform.position.y > transform.position.y)
    {
        playerRigidBody.velocity = new Vector2(playerRigidBody.velocity.x, -playerHurtForce);
    }
    else
    {
        playerRigidBody.velocity = new Vector2(playerRigidBody.velocity.x, playerHurtForce);
    }
}
```

Ilustración 13: Código para la colisión de los obstaculos con el jugador

Cuando el jugador entra en contacto con un objeto de tipo obstáculo, recibe daño, se desplaza ligeramente en la dirección opuesta y se reproduce la animación que corresponde con el estado de *Herido*.

Además de compartir esta funcionalidad, *Mace* también se desplaza por el escenario, de un punto a otro.

```
if (transform.position.x > leftCheckPoint.position.x)
{
    obstacleRigidBody.velocity = new Vector2(-obstacleHorizontalSpeed, obstacleVerticalSpeed);
}
else
{
    endReached = true;
}
```

Ilustración 15: Desplazamiento del objeto Mace

Ilustración 16: Código de la clase GameMaster

Este fragmento de código forma parte del método que mueve el obstáculo, y lo que hace es comparar si éste se encuentra entre los dos puntos y darle velocidad negativa o positiva, en función de la dirección.

GameMaster y la persistencia de la información

En unity, cada vez que se carga una nueva escena o se

vuelve a cargar la misma, todos los objetos y sus respectivas instancias se eliminan y se vuelven a crear, haciendo que su información desaparezca.

Esto es extremadamente útil, ya que así se puede repetir y rejugar los mismos escenarios o niveles, pero también puede acarrear algunos problemas.

Por ejemplo, si el Player tiene 3 vidas como máximo y ya ha perdido una, al a cargar de nuevo el escenario, su cantidad de vidas volvería a ser 3. Esto supone un gran inconveniente, ya que si se ha de volver a recargar la escena aún a pesar de no haber terminado el nivel, por ejemplo, si el jugador se ha caído del escenario, éste volvería a tener todas sus vidas, además de perder toda la información de la escena.

Para este tipo de casos, existe la función *DontDestroyOnLoad*, un método de Unity que no destruye el objeto cuando se recarga la escena. Aún así, no es tan sencillo como simplemente marcar los GO con esta función para que no sean eliminados, ya que tan solo se busca la persistencia de cierta información de algunos objetos, no de toda la instancia.

Por ello, se utilizará un patrón de programación llamado *Singleton*, que consiste en crear una clase -a la que se llamará GameMaster- que será similar a un contenedor de datos; los objetos de GameMaster no serán destruidos y serán accedidos por otras instancias que necesiten persistir sus datos. A continuación, el código que le corresponde.

En las primeras líneas, están las variables que guardarán la información que se quiere persistir, y en la función Awake() -que se ejecuta cuando se crea el objeto GameMaster-, se encuentra el código que impedirá la destrucción de este objeto.

Por último, lo único que ha de hacerse es llamarse a GameMaster cada vez que quiera actualizarse la información y cuando se necesite destruir, como por ejemplo, al terminar el juego.

```
private static GameMaster instance;
public Vector2 lastCheckPointPosition;
public int playerLifes = 3;
Mensaje de Unity | 0 referencias
private void Awake()
{
    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(this);
    }
    else
        Destroy(gameObject);
}
```

SceneLimit y TheEnd

SceneLimit es un GO muy sencillo, que consiste en un *Collider* y en una Script muy simple, y se encarga de detectar si el jugador se ha caído del scenario. En caso de ser así, llama a la función de *PlayerController* encargada de quitarle una vida y de comprar si le quedan vidas o no. En caso de que sea así, vuelve a cargar el escenario, pero en caso contrario, carga la pantalla de *Game Over*.

Por el contrario, TheEnd es un objeto situado el final del escenario, y se trata de la meta que debe alcanzador el jugador, es el encargado de determinar que Player ha llegado al final y, por tanto, ha terminado el nivel.

Consiste de un Sprite -una imagen en forma de casa-, un *Collider* y un sencillo Script.

Cuando el jugador entra en contacto con la casa, se carga una pantalla que nos indica que hemos superado este escenario.

Otras escenas: menús y pantallas

Los elementos mencionados en los apartados anteriores son algunos de los componentes principales de las escenas creadas para montar los escenarios del juego, pero hay otro tipo de escenas en este juego: los menús y las pantallas.

Por un lado, está el menú principal, o *MainMenu*, el cual tiene el siguiente aspecto

Ilustración 17: Escena del menú principal

De la misma manera que otras interfaces como PlayerUI y PauseMenuUI, este menú consta de un Canvas en el cual se han añadido diversos elementos gráficos.

Entre las opciones que se pueden escoger, todas han sido ya explicadas en apartados anteriores, menos "Jugar". Al pulsar este botón, el juego selecciona de forma aleatoria cualquiera de los escenarios implementados y lo ejecuta inmediatamente.

Ilustración 18: Menú de opciones

En el menú de opciones, también comentado anteriormente, se pueden configurar la calidad de imagen, la resolución, el volumen y el tamaño de la ventana del juego

5. MODELO DE NEGOCIO

Una vez se definió la idea del proyecto -tanto sus características, como algunas de las herramientas que se necesitaban para su desarrollo-, el próximo paso que se dio fue determinar el modelo de negocio.

En el caso de la industria de los videojuegos, hay una serie de arquetipos clásicos, que la mayoría (sino todos) los juegos suelen seguir, y son los siguientes:

- Modelo *Pay-to-Play*: Consiste en hacer pagar el contenido y el uso del videojuego, normalmente solo se suele pagar una vez por el juego (la primera) y no hay limitaciones en su uso. Es lo más parecido a la compra convencional: se compra el producto (juego) y se puede usar sin restricciones.

- Modelo *Free-to-Play*: Consiste en ofrecer gratis el contenido y el uso del videojuego. En este modelo la monetización proviene de la publicidad y/o la venta de bienes/servicios virtuales.

- Modelo *Freemium*: Consiste en ofrecer gratis el uso del videojuego, pero no la totalidad de su contenido, al cual se accede pagando.

- Modelo *Suscripción*: Consiste en hacer pagar el contenido y el uso del videojuego de forma periódica. El uso de éste está limitado, y es similar a la suscripción de un servicio, puede utilizarse mientras se sigue suscrito.

Estos paradigmas de negocio son los más típicos, aunque también se pueden encontrar modelos híbridos que combinan varios de estos conceptos como, por ejemplo, comprar el juego y luego tener que pagar o bien una suscripción por su uso o bien una serie de micro transacciones a cambio de contenido adicional.

En el caso de este proyecto, se ha optado por un

modelo *Pay-to-Play* para la versión de PC y de consolas, y una versión *Free-to-Play* para las plataformas *mobile*. Además, contaría con contenido de pago, como nuevos niveles o personajes.

Para terminar de concretar la idea de negocio entorno a *Evolving*, se ha realizado un *Bussiness Model Canvas*, que es una plantilla que permite gestionar y visualizar de forma gráfica algunos de los aspectos principales de todo nuevo negocio. En el caso de este proyecto, se ha utilizado el *Full Bussiness Model Canvas* (FBMC), que es una modificación más detallada del esquema clásico, realizada por Juanjo Villanueva, el tutor de este TFG.

En el apartado 8.3 de los anexos, tenemos el FBMC del proyecto *Evolving*.

6. Conclusión

Para el desarrollo de este proyecto se ha realizado una metodología ágil como SCRUM, en la que se han realizado reuniones semanales con el tutor. En estas se presentaba el trabajo semanal realizado, y se ponían nuevos -o los mismos, en caso de no haber finalizado el trabajo de la iteración anterior- objetivos a realizar para la próxima semana.

Durante el desarrollo de este proyecto no se ha podido llevar el ritmo de trabajo y la eficiencia deseadas, y esto ha hecho que la obtención de resultados haya sido más la lenta de lo planificado.

Finalmente, se ha conseguido llegar a los objetivos y cumplir los requisitos básicos, ya que se ha podido implementar el personaje principal y sus mecánicas, además de tres tipologías de enemigos, 3 niveles, un menú principal y un menú interno del juego.

Como proyección de futuro para este proyecto, sería interesante poder implementar la mecánica de evolución de forma más completa, ya que ahora solo aumenta de forma muy sencilla algunas características de los enemigos. Además, también se podrían añadir más tipos de enemigos, con movimientos y una inteligencia artificial más compleja, así como más niveles y obstáculos.

7. Bibliografia

[1] Unity Unity Real-Time Development Platform | 3D, 2D VR & AR Engine

[2] TechGuided – Best Game Engines 2020 7 Best Game Engines in 2020 (Free, Graphics, Beginners, 2D, 3D) (techguided.com)

[3] Unity Documentacion Unity - Scripting API: (unity3d.com)

[4] Wikipedia – List of game engines Unity - Scripting API: (unity3d.com)

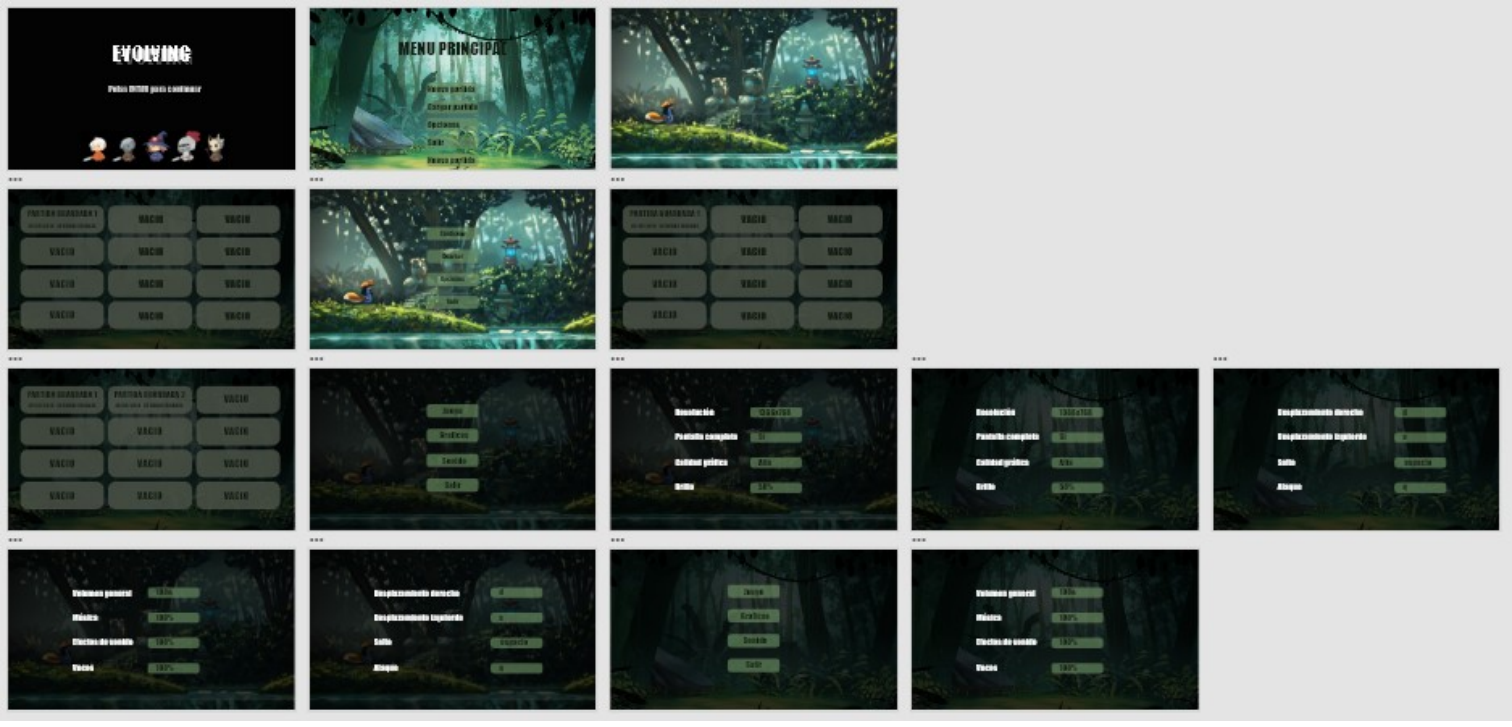
8.

9. ANEXOS

9.1. Tabla comparativa de motores gráficos

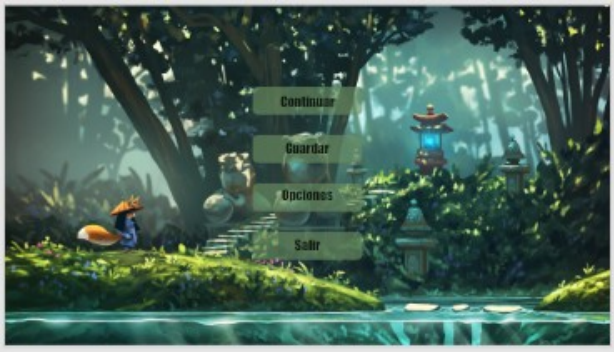
	Unity	Unreal Engine	GameMaker	CryEngine	Amazon Lumberyard
Precio de utilización	Gratuito. Su versión gratuita incluye todas las funcionalidades principales, el *core*, pero se puede pagar una mensualidad por planes profesionales que incluyen funcionalidades extra[1].	Gratuito, pero una vez se lance el juego y se hayan superado los primeros 3000$ de ganancias, se le ha de pagar a la compañía un 5% de todo lo generado.	Gratuito, pero para poder sacarle el máximo partido, es necesario comprar la las versiones *Professional* o *Master Collection*.	Gratuito, pero una vez se lance el juego y se hayan superado los primeros 5000$ de ganancias, se le ha de pagar a la compañía un 5% de todo lo generado.	Gratuito, pero si se quiere que el juego tenga un componente online y multijugador, se ha de pagar un plan Amazon Web Services.
Dificultad de aprendizaje	Media.	Alta.	Baja.	Alta.	Media.
Complejidad en su uso	Media.	Alta.	Baja.	Alta.	Media.
Lenguaje utilizado	C#.	C++.	Ninguno.	C++, C#.	Lua.
Documentación	Dispone de una gran documentación, tutoriales, cursos, foro, etc.; tanto propia como externa. En este apartado es el que más destaca de los cinco.	Dispone de una gran cantidad.	Tiene una cantidad aceptable.	Tiene una cantidad aceptable.	Tiene, pero en este aspecto escasea un poco.
Potencia gráfica	Media/alta.	Alta/muy alta.	Baja/media.	Alta/muy alta.	Media.
Multiplataforma	Sí.	Sí.	Sí.	Sí.	Sí.
Orientado a 2D o 3D	Ambas, aunque destaca notablemente en 2D.	Ambas, aunque destaca notablemente en 3D.	Ambas.	3D.	3D.
Librerías, *tools*, *assets*	Muchísima disponibilidad, incluida las herramientas y *assets* de la comunidad de usuarios.	Disponibilidad muy alta.	Disponibilidad, aunque no tan abundante como el resto.	Disponibilidad aceptable.	Disponibilidad, aunque no tan abundante como el resto.

8.2. Prototipo: Maqueta interactiva

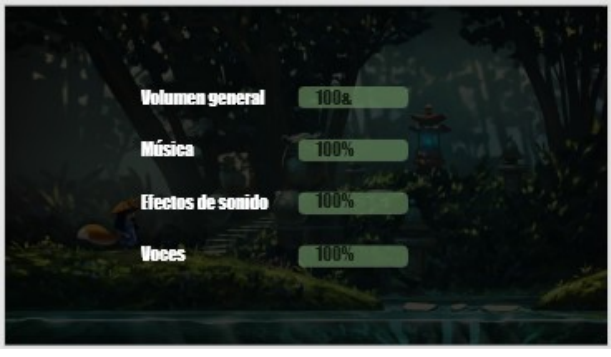


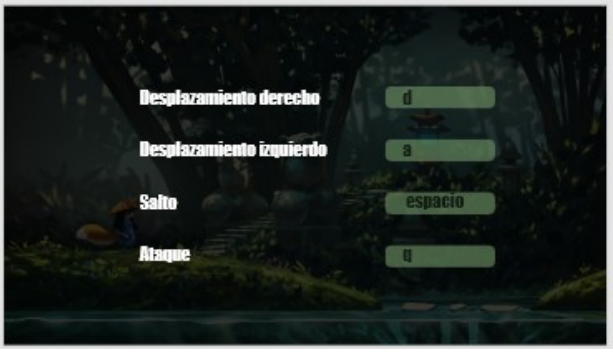

EVOLVING
Pulsa ENTER para continuar

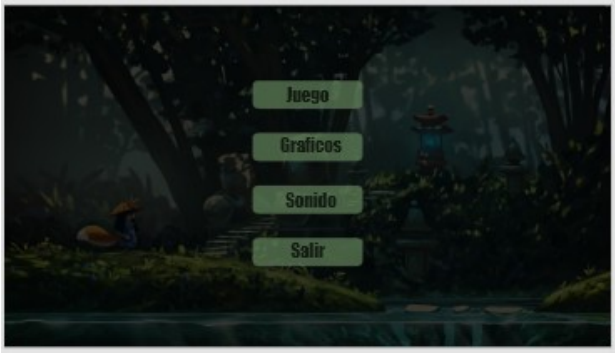

MENU PRINCIPAL
Nueva partida
Cargar partida
Opciones
Salir
Nueva partida


Continuar
Guardar
Opciones
Salir


PARTIDA GUARDADA 1
VACIO
VACIO
VACIO
VACIO
VACIO
VACIO
VACIO
VACIO
VACIO


Volumen general 100&
Música 100%
Efectos de sonido 100%
Voces 100%


Desplazamiento derecho d
Desplazamiento izquierdo a
Salto espacio
Ataque q


Juego
Graficos
Sonido
Salir


Resolución 1366x768
Pantalla completa Si
Calidad gráfica Alta
Brillo 50%

8.3. Full Business Canvas

<table>
<tr><td>Proyecto: Evolving
Diseñado por: Jordi Farré Delgado
Fecha:
Iteración: 2</td><td colspan="3">0 Idea de Negocio: Un videojuego -Evolving-, cuya característica principal sea la evolución y la adaptación: los enemigos cada vez se van haciendo más fuertes y mejoran sus estrategias para defenderse contra el jugador y derrotarle; eso incluye hacerse más resistentes a los ataques más utilizados, cambiar sus patrones de ataque en función de combates anteriores y de su estilo de juego, etc. Asimismo, los niveles también cambiarían, poniendo más obstáculos y variando en función de como se hayan completado los niveles anteriores.</td></tr>
<tr><td>13 Fortalezas</td><td>9 Ingresos</td><td>2 Propuestas de Valor</td><td>1 Clientes y Usuarios</td></tr>
<tr><td>Mecánicas originales

Intención de actualizarlo con frecuencia para que el usuario no se aburra

Juego con una dificultad accesible y que genera sensación de logro

Fácil de jugar en cualquier sitio y momento</td><td>Versión base gratuita para la plataforma mobile

Versión de pago en el resto de plataformas: PC y consolas

Posibles expansiones o niveles adicionales de pago

Contenido extra, como más personajes o ítems, disponibles mediante micropagos</td><td>Juego de plataformas divertido y entretenido

Dificultad adaptativa al usuario, de manera que no sea ni demasiado fácil ni muy frustrante

Mecánicas interesantes para el usuario, como la de la evolución del personaje y enemigos</td><td>Jugadores asiduos de cualquiera de las plataformas

Gente que juega de forma puntual: trayectos de transporte público, tiempos de espera

Jugadores interesados por los juegos 2D

Jugadores interesados por los juegos de plataformas</td></tr>
<tr><td>14 Debilidades</td><td>10 Costes</td><td>4 Soluciones y mejoras</td><td>3 Problemas y Deseos</td></tr>
<tr><td>Mucha competencia; el mercado de los videojuegos es muy amplio y tiene muchos productos para el usuario

Ya existen muchos juegos de plataformas

Actualmente, los géneros más populares son otros, como los MOBA o los FPS

En la etapa inicial, el juego será una versión muy básica de sí mismo

Al principio habrá pocos ingresos</td><td>Formaciones para futuros empleados

Sueldo de los trabajadores (en esta etapa del proyecto, es 0)

Costes de diseño e infraestructura, como servidores, clouds, bases de datos</td><td>Fuente de ocio que pretende entretener al usuario

Se puede jugar en periodos cortos de tiempo gracias a tener niveles cortos y a tener una versión mobile

El contenido base del juego será gratuito (versión mobile) o a un precio muy asequible</td><td>Una parte amplia de nuestros clientes son gente joven: estudiantes o trabajadores

Los usuarios disponen de poco tiempo

Un sector amplio de los usuarios no tienen una fuente estable de ingresos

Se aburren rápido</td></tr>
</table>

15 Oportunidades	**11 Métricas**	**6 Canales**	**5 Mercado**
Aprovechar la poco explotada mecánica principal del juego, diferenciando a *Evolving* de otros juegos de plataformas Es un juego nuevo que no pertenece a ninguna saga, por tanto el usuario no tiene ninguna opinión todavía: ni positiva ni negativa La gente asidua a jugar suele informarse de los nuevos lanzamientos y están abiertos a probarlos	Número de unidades vendidas o descargadas del juego base, tanto en total como por plataforma Ingresos por la venta del juego Ingresos por contenidos de pago y micropagos Valoración de los usuarios en plataformas como Steam Valoración de expertos en revistas y artículos	Steam Android Store Youtube, Twitch Instagram, Twitter, Facebook	Países sin restricciones legales que impidan este tipo de producto Segmento especifico del mercado: jugadores o gente que disfruta de los juegos, en general Usuarios mayores de 7 años (edad recomendada)
16 Amenazas	**12 Hitos**	**8 Recursos**	**7 Competencia**
Que *Evolving* no genere una primera impresión lo suficientemente atractiva como para que tenga una gran cantidad de usuarios Otros juegos similares Que el producto inicial sea demasiado distinto a como pretende ser en el futuro, y los usuarios dejen de jugar antes de tiempo Otras Start-Ups o empresas con ideas parecidas	Entrega de la fase beta y finalización del TFG Fecha de salida Fecha de actualización Adición de contenido extra	El creador principal del proyecto Un desarrollador Un PC preparado para el desarrollo del juego Motor gráfico, Unity Assets de la comunidad de Unity Otros Softwares, como los ofimáticos o los de diseño	La saga de Super Mario La saga de Ori Ratchet & Clanck Trilogía de Crash Bandicoot