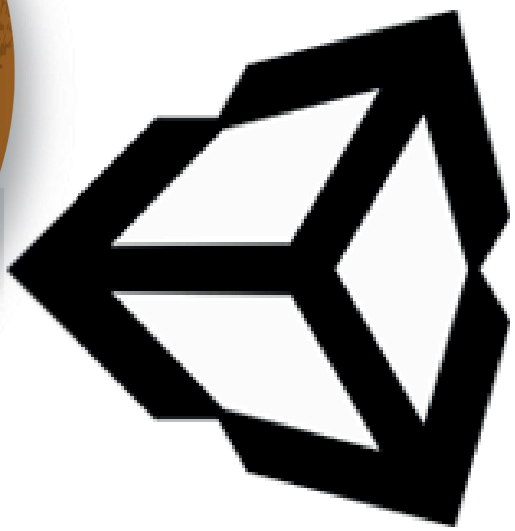


UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Creación de un juego 2D con Unity

Universitat Politècnica de València
Campus d'Alcoi

Grado en Ingeniería Informática

Trabajo de fin de grado

Memoria presentada por: Javier Pardo Sanz
Tutor: Jordi Joan Linares Pellicer

Convocatoria de defensa: Septiembre 2017

Índice

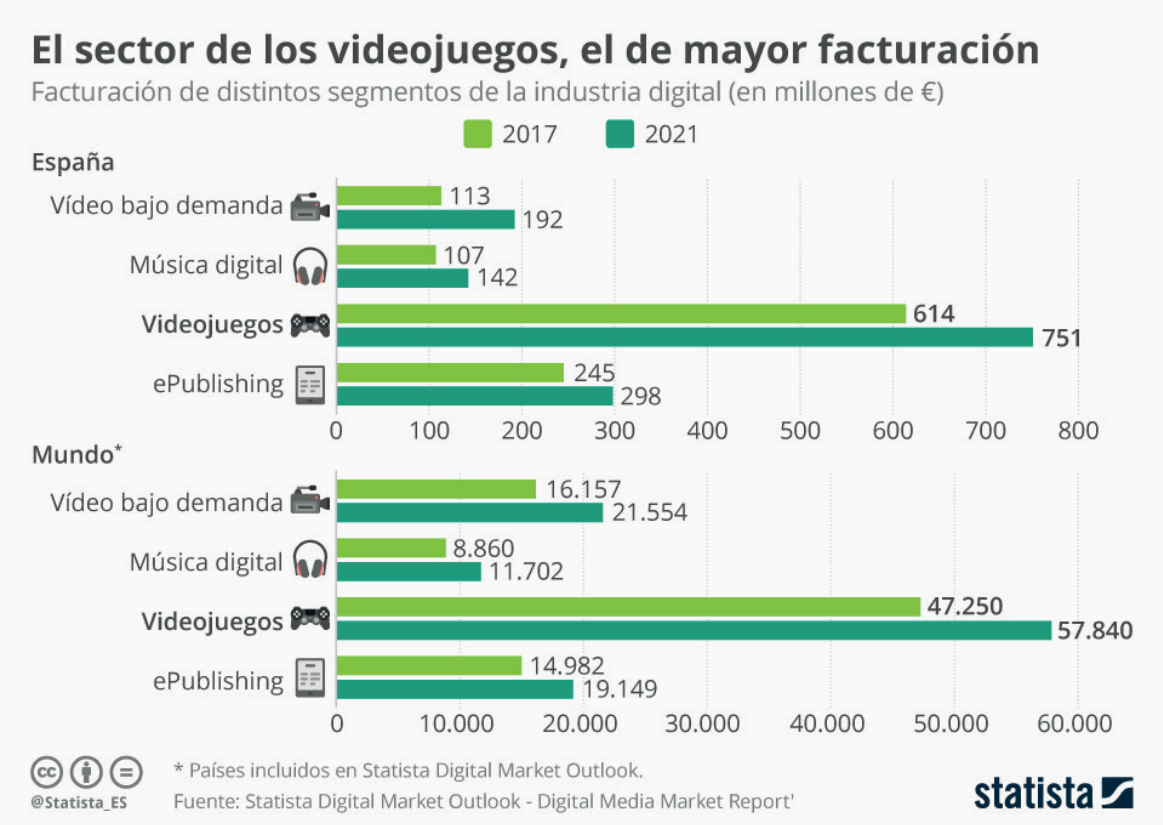
1-Introducción

1.1Motivación

Mi motivación para este proyecto junto al hecho de crear mi primer videojuego, es el interés de aumentar tanto mis conocimientos sobre el desarrollo de videojuegos, además de conseguir experiencia laboral en este sector y la pasión de esta nueva forma de entretenimiento, cambiar de un punto de vista como jugador a desarrollador de videojuegos.

1.2 Trasfondo del Trabajo de Fin de Grado

En estos últimos años la industria de los videojuegos se ha llegado a convertir en una gran potencia en el mundo, llegando superar tanto la industria de la música, como la del cine.
Esto se debe en parte al desarrollo de nuevas tecnologías y a la reducción de barreras para que los usuarios puedan crear sus propias IPs con una mayor calidad, con un número más limitado de recursos. Algunas de estas herramientas son Unity o Unreal Engine.



1-Introducción

El mundo de los videojuegos ha llegado a abarcar una gran cantidad de géneros y subgéneros, se ha llegado a punto de que muchos videojuegos son considerados obras de arte, donde algunos estos no tienen un simple objetivo de entretener, sino que también nos narran sus historias, transmiten experiencias, etc. Alguno de estos juegos son: el Super Mario, Final Fantasy VII, The Witcher III, Limbo, entre otros.

También se desarrollan videojuegos como medio educativo, tanto para aprender matemáticas y lógica. Otros no son tan enfocados a todos los públicos, como algunos simuladores de aviones, donde estos se utilizan para enseñar a los pilotos. Además de estos también existen juegos accesibles a discapacitados, donde los desarrolladores que se especializan en este sector adaptan los juegos para que las personas con discapacidad puedan disfrutarlos. Un ejemplo en este campo seria Final Fantasy XIV: A Realm Reborn que fue premiado como el título más accesible en 2013 , donde adaptan los colores para todos los usuarios, incluyendo los usuarios daltónicos.

En la actualidad, gracias a los motores gratuitos de videojuegos, salen una gran cantidad de videojuegos al año, esto hace que se clasifiquen según la empresa que desarrolla el videojuego.

Se pueden clasificar en:

- Grandes empresas.
- Empresas independientes.

Las clasificadas como grandes empresas, me refiero a empresas como Nintendo, Microsoft Studios, CD Projekt, etc.

Son las empresas que normalmente hacen los juegos clasificados como “triple-A”. Los juegos que reciben esta denominación, son en los que se invierten grandes cantidades de dinero y poseen grandes recursos para poder dedicarle que salga lo mejor posible.

1-Introducción

Algunos ejemplos de títulos triple-A son:

The Witcher III

Final Fantasy (Cualquier entrega de la saga)

Mario Odyssey

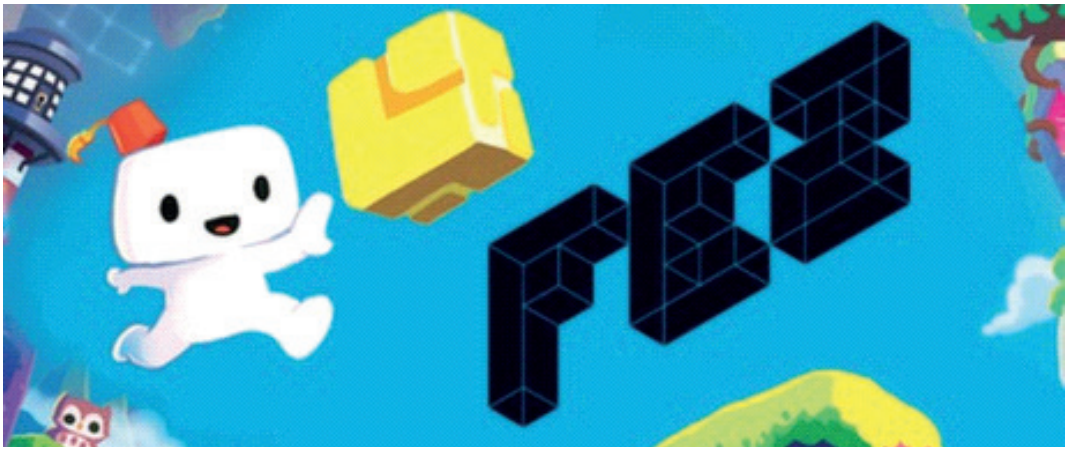
1-Introducción

A diferencia de estas grandes empresas, hay otras que no poseen ni esas cantidades de dinero, ni esos recursos. A estas empresas se les conoce como empresas independientes o “indies”.

La empresas indies son pequeñas empresas, que no poseen grandes recursos, que optan en su mayoria por un enfoque distinto en sus proyectos. Algunas de las empresas indies son Tequila Works, Lince Works, Polytron Corporation , Nexus Game Studios, etc.

Unos ejemplos de títulos indies son:

Limbo



Fez

1-Introducción

Aragami

Randal’s Monday

Los juegos Aragami y Randal’s Monday han sido creados por desarrolladores españoles.

1.3-Objetivos

Crear un juego 2D desde cero, usando el motor de Unity. Este juego tendrá varios géneros como el beat’em up (en 2D) y plataformas. En un principio esta pensado para la plataforma de Pc, pero no se descartan que se incluyan otras plataformas en un futuro.

Los géneros mencionado antes son lo siguiente:

Beat ‘em up: Subgénero de acción de ir derrotando oleadas de enemigos que se te cruzan por el camino mientras avanzas.

Plataformas: Subgénero de aventuras donde el jugador controla a un personaje que debe avanzar por el escenario evitando obstáculos físicos, ya sea saltando, escalando o agachándose.

2-Descripción de las herramientas utilizadas

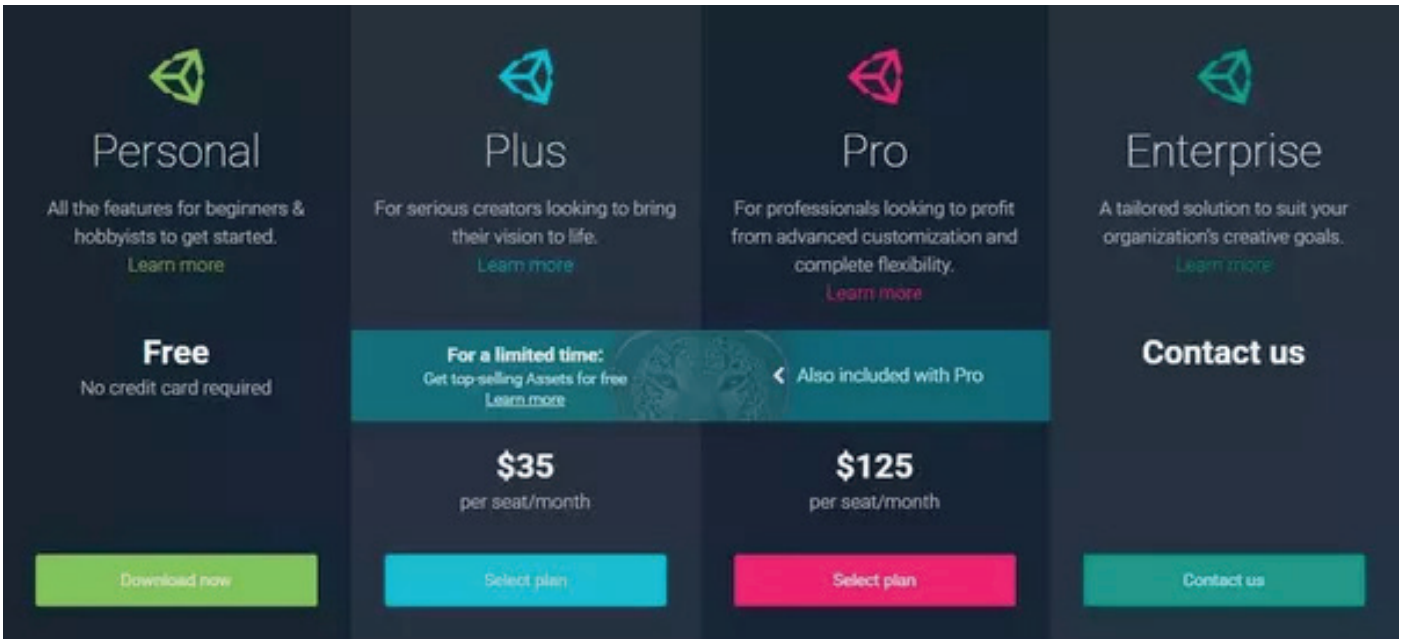
En la actualidad hay gran cantidad de programas que ayudan a la hora de hacer videojuegos y cada uno con sus características. Los utilizados de momento en este proyecto son los siguientes:

-Unity
-Mono Development
-FL Studio
-Adobe Photoshop
-Sai
-Spine

2.1-Unity

Unity es un motor de videojuegos multiplataforma creado por Unity Technologies. Unity puede desarrollar juegos tanto en 2D como en 3D, los lenguajes con los que se puede trabajar son C Sharp y Java Script y las plataformas en las que puede desarrollar son las de Pc (Windows, Linux, Mac OS), móviles (Windows, Android, iOS), consolas (Playstation, Xbox, Nintendo), web(WebGL), entre otras.

Anteriormente unity tenia dos versiones, pero actualmente consta de 4 versiones:



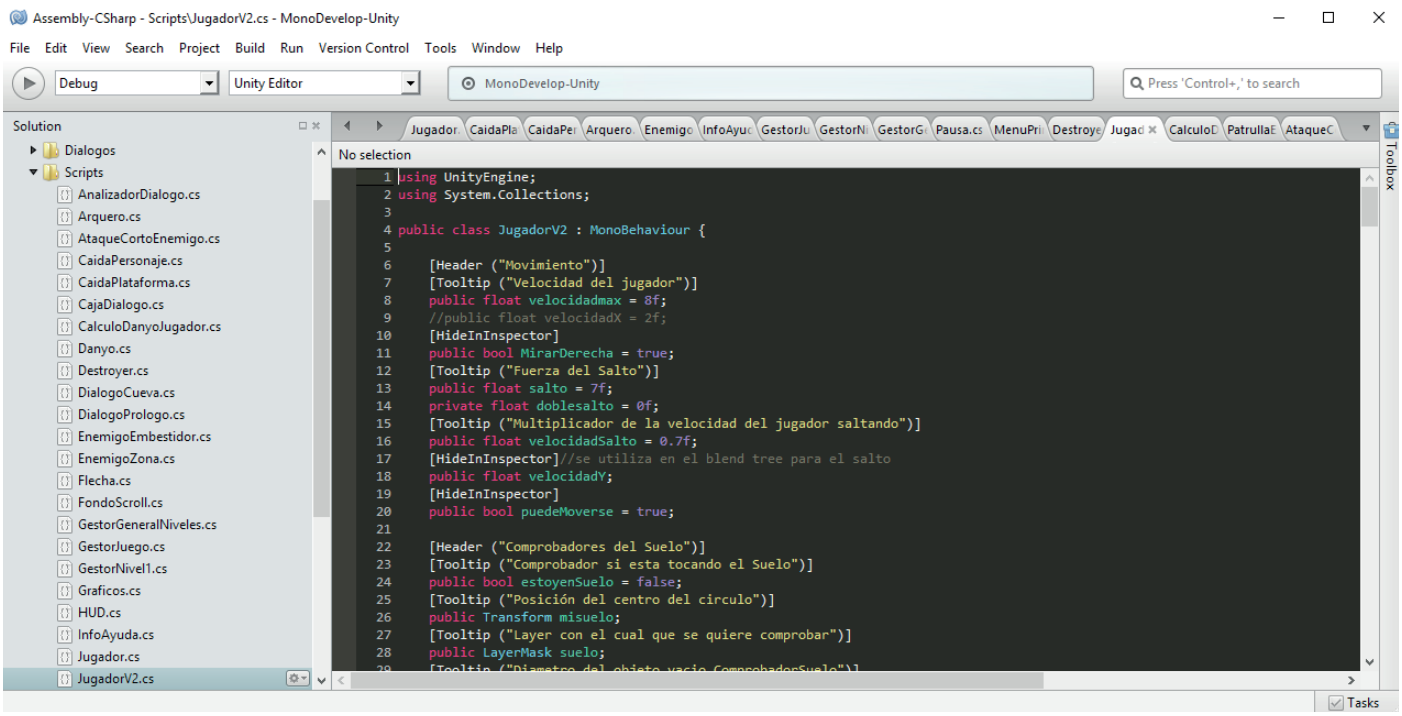
2-Descripción de las herramientas utilizadas

La mayor diferencia entre estas versiones, sin contar de que las versión Personal es algo más limitada que las otras, es la monetización al adquirir una de las versiones. Las versiones Personal y Plus tienen un limite de 100,000$ y 200.000$ respectivamente. La versión Pro y Entrerprise ya no poseen esta limitación, además de tener unas mejoras a la versión Plus, como mayores descuentos, etc.

2.2-Mono Development

Mono Development es un entorno de desarrollo integrado libre y gratuito.

A la hora de programar scripts en unity, este nos da la opción de instalar además del Mono Development el Visual Studio, pero con las características que posee el dispositivo con el cual se ha programado este proyecto, opte por Mono Development ya que gastaba menos recurso que Visual Studio.



2-Descripción de las herramientas utilizadas

2.3-Fl Studio

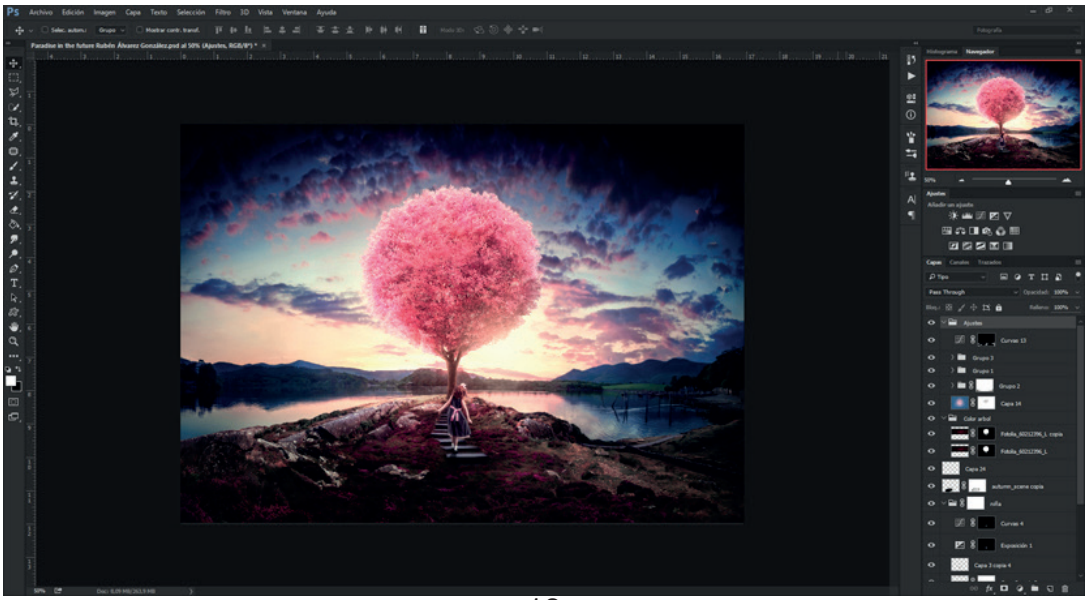
Fl Studio (anteriormente conocido como Fruity Loops), es una estación de trabajo de audio digital con las características de editor de audio, etc.

La versión de demostración de Fl Studio tiene varios sintetizadores de software y algunos de estos deben ser adquiridos por separado. Este programa se esta utilizando para la banda sonora.

2.4-Adobe Photoshop

Adobe Photoshop conocida simplemente como Photoshop es un editor de gráficos rasterizados desarrollado por Adobe System Incorporated.

Photoshop es un programa de pago que trabaja con capas y se ha utilizado para el diseño de los escenarios.

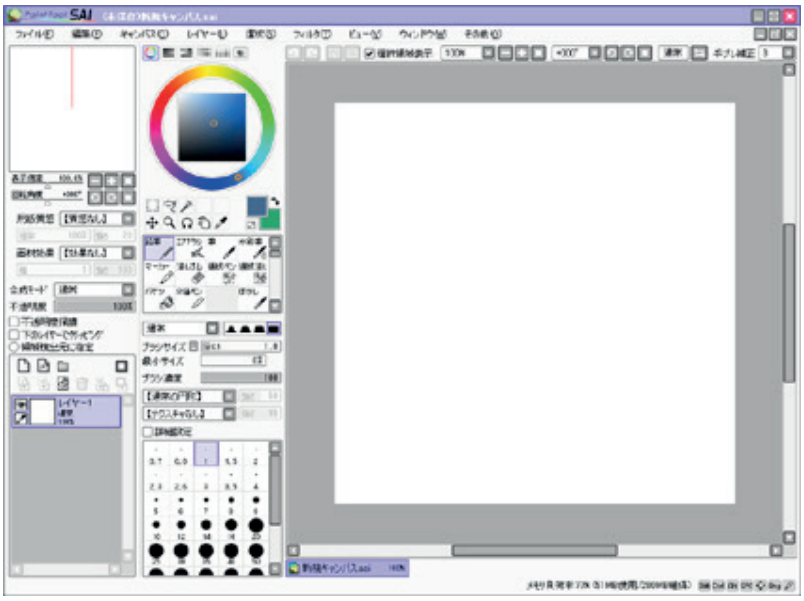


2-Descripción de las herramientas utilizadas

2.5-Sai

Easy Paint Tool Sai o simplemente Sai, es un software de ilustración para Microsoft Windows, desarrollado por SYSTEMAX. Sai es un programa de pago también, que el valor de su licencia es de 5.400 yenes que equivalen a 41,47 ,pero a diferencia de Photoshop no es de pago mensual.

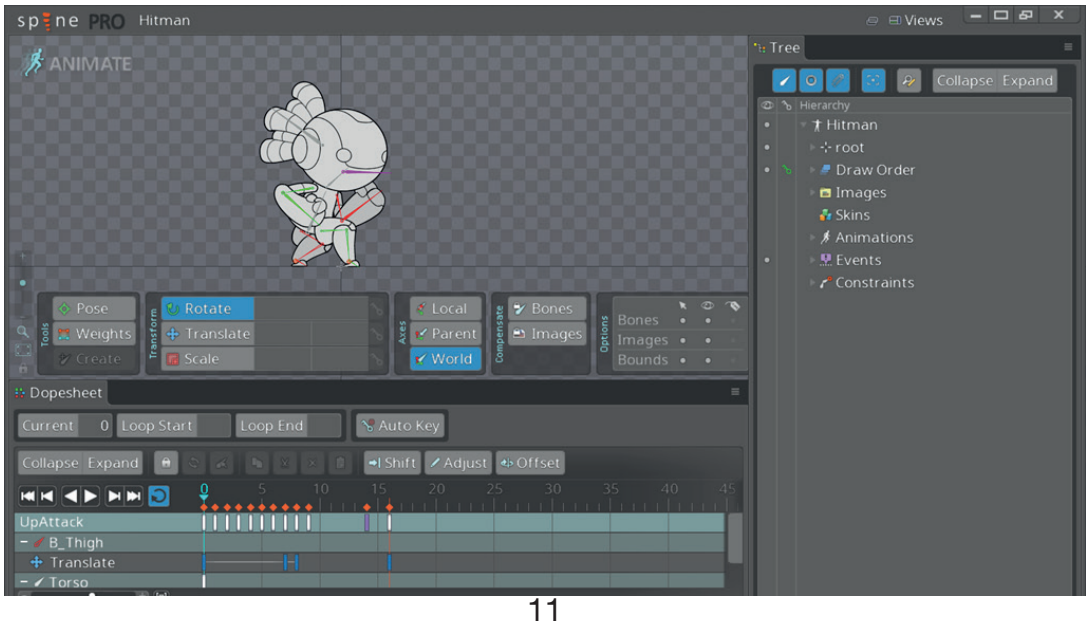
Sai es una aplicación ligera, que trabaja con capas al igual que photoshop, pero se ha utilizado para los personajes porque ha resultado más sencillo colorearlos que con photoshop.



2.6-Spine

Spine es un software de animación 2D desarrollado por Esoteric Software, donde proporciona varias herramientas para modelar y refinar las animaciones. Spine también esta preparado para exportar los datos de las animaciones a Unity.

La licencia de este software son 299$.



3-Diseño y desarrollo de la aplicación

3.1-Entorno de Unity

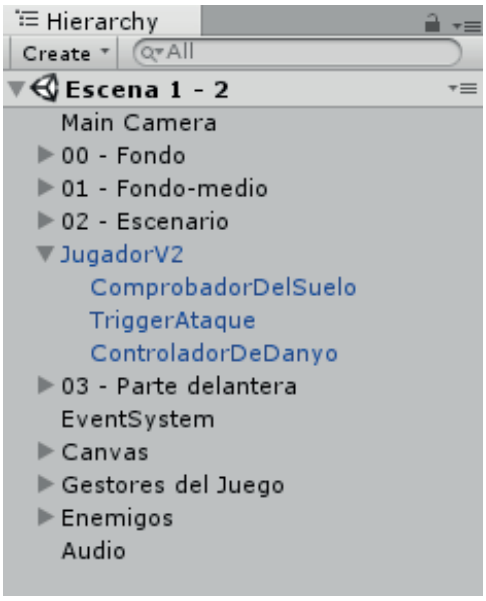
Como se ha mencionado en el punto anterior Unity es un motor de videojuegos que puede hacer tanto juegos en 2D como 3D.

Cuanto a sus características, todos los objetos que interaccionan en el juego son conocidos como GameObjects, ya sea la cámara, el jugador o un objeto vacío. Estos objetos estarán en una escena, que sería cualquier nivel del juego. Al cambiar de escenas, a no ser que especifique en una script se eliminarían los objetos de la escena anterior. También se puede crear el denominado prefab, que es un objeto con unas características especificas que una vez convertido se transforma en una especie de plantilla, el cuál gracias a los prefabs puedes crear objetos idénticos en otras escenas.

El editor de Unity se compone de varias ventanas que de agrupan en pestañas como la de escena, la de juego, la de jerarquía, la de proyecto, la consola, la animación, el animador, el asset store, entre otros.

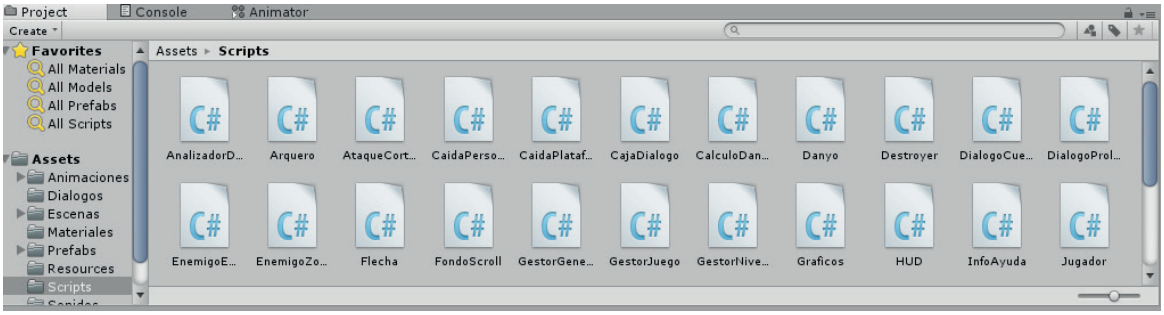
Las pestañas más significativas se podrían decir que son las de escena y el juego ya que son donde se colocarán los objetos en la escala, posición y rotación que se deseen y con la del juego podremos ver como se vería en la pantalla del juego realmente con las proporciones de la pantalla respectivamente.

3-Diseño y desarrollo de la aplicación

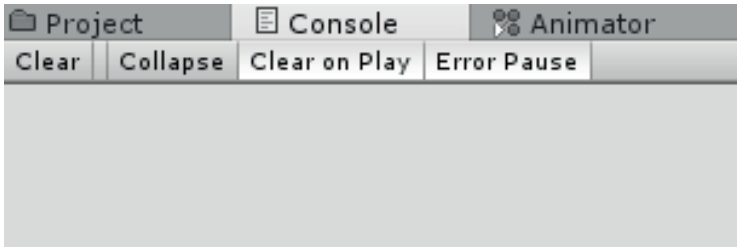


La pestaña de la jerarquía muestra los objetos que se tiene en la escena y sus propiedades ya sea escala, rotación, posición, capa entre otros. Esta pestaña actúa junto al inspector para poder modificar las características dichas anteriormente.

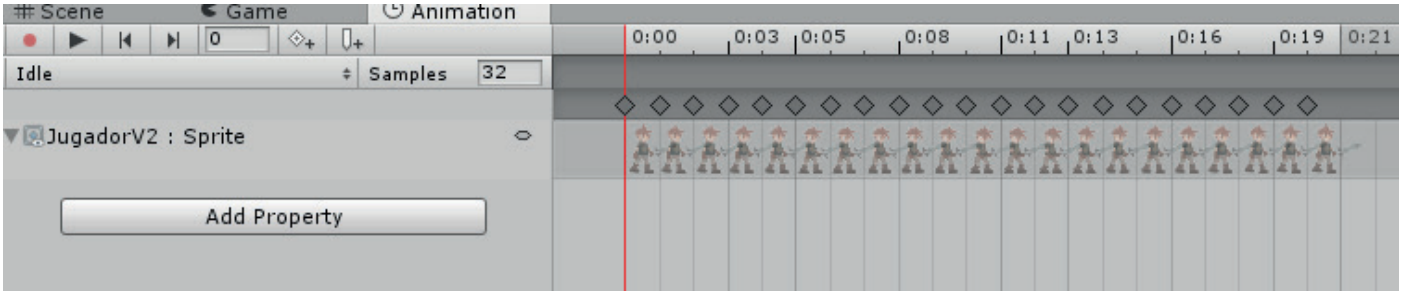
La pestaña del proyecto muestra la disposición de las carpetas y archivos del que se compone el proyecto pudiendo acceder a estos y modificarlos con mayor facilidad.



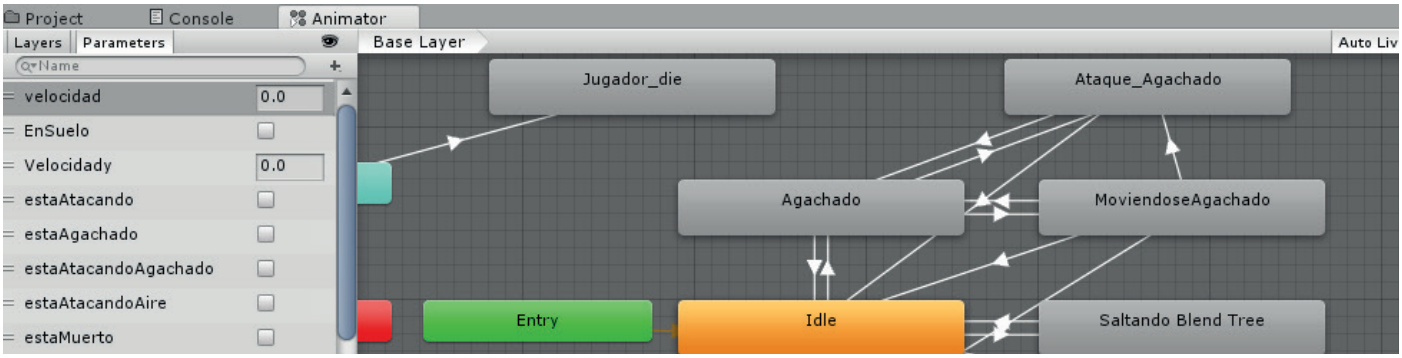
La pestaña de la consola también es de las más significativas, el cual muestra las advertencias, los errores y los mensajes de debug que se activen por alguna script.



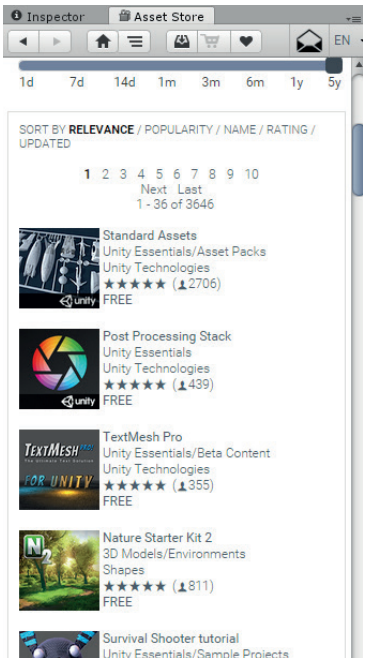
3-Diseño y desarrollo de la aplicación



Las pestañas de animación y animador son las que crean las animaciones y la que configuran las animaciones respectivamente.



Por último, la pestaña Asset Store, es la tienda oficial de Unity donde se pueden comprar varios recursos de Unity, ya sean sonidos, scripts, etc.



3-Diseño y desarrollo de la aplicación

3.2-Principales retos tecnológicos

Han habido varios retos en el proceso de este proyecto. El primero es el personaje principal, ya que es uno de los puntos más importantes o el más importante de todo el proyecto, ya que no solo es programar los movimientos y ataques. El mayor problema que ha representado es a la hora de programar los pequeños detalles, como que cuando ataque no se mueva para que concuerde con la animación, el cambio de velocidades cuando esta corriendo o saltando, o la animación cuando traspasa las plataformas.

Otro gran reto fueron los diálogos, cuando lo configure al principio con una general para que solo cambiando el .txt cambiara el diálogo, pero al probarlo en el ejecutable no salía el diálogo. Lo resolví con una script nueva para cada diálogo, con esto conseguí ventajas y desventajas. Las desventajas son que tenía que hacer una script para cada diálogo. Las ventajas es que se puede personalizar cada diálogo ya sea el texto las imágenes o las posiciones.

Y por último las inteligencias artificiales de los enemigos, donde cuanto más complejo es el enemigo, más condiciones tienen y es bastante más difícil programar a este correctamente.

3.3-Características y arquitectura de la aplicación

El juego tiene varias escenas que lo componen. Un menú donde se puede acceder a las opciones, créditos, selección del nivel o salir del juego. No esta configurado que cuando pulses cualquier botón del menú cambie de escena, sino que activa y desactiva los denominados Canvas para mostrar cada sección en la misma escena.

Todas las escenas tienen como mínimo un objeto Gestor, el cual posee una script que no se destruye al cambiar de escena y que posee varias variables donde comprobamos si el jugador esta vivo, cuantas monedas tiene, etc.

Una vez seleccionado un nivel en la escena se muestra el gestor mencionado anteriormente, además de uno para este nivel especifico. También se muestra el protagonista y sus enemigos junto con el escenario, donde este esta formado por varias capas, los cuales le darán profundidad además de varios objetos como plataformas o elevaciones. También hay un objeto audio para la banda sonora y varios canvas para la pausa, el gameover y los mensajes del juego.

3-Diseño y desarrollo de la aplicación

3.4-Descripción de los elementos tecnológicos más característicos del juego

3.4.1-Interfaz

En la interfaz de Unity se utilizan los objetos denominados canvas, a este objeto se le pueden asignar imágenes, botones, textos, etc.

En cada nivel o escena se utiliza este objeto para diversas cosas. En el menú principal y final se les asigna los botones para cambiar de escena o cambiar la configuración.

Para las escenas de diálogos, es para crear el diálogo en sí, las imágenes, las cajas de diálogo y los textos.

Y en las escenas jugables son la vida en forma de corazones, el menú de pausa, el gameover entre otros. Cada canvas tendrá sus características propias, ya sean mensajes para informar, botones para reiniciar nivel, para comprar objetos, etc.

3.4.2-Niveles

El proyecto, al ser una demostración técnica se compondrá de 3 niveles, los cuales cada nivel se compondrá de 2 escenas.

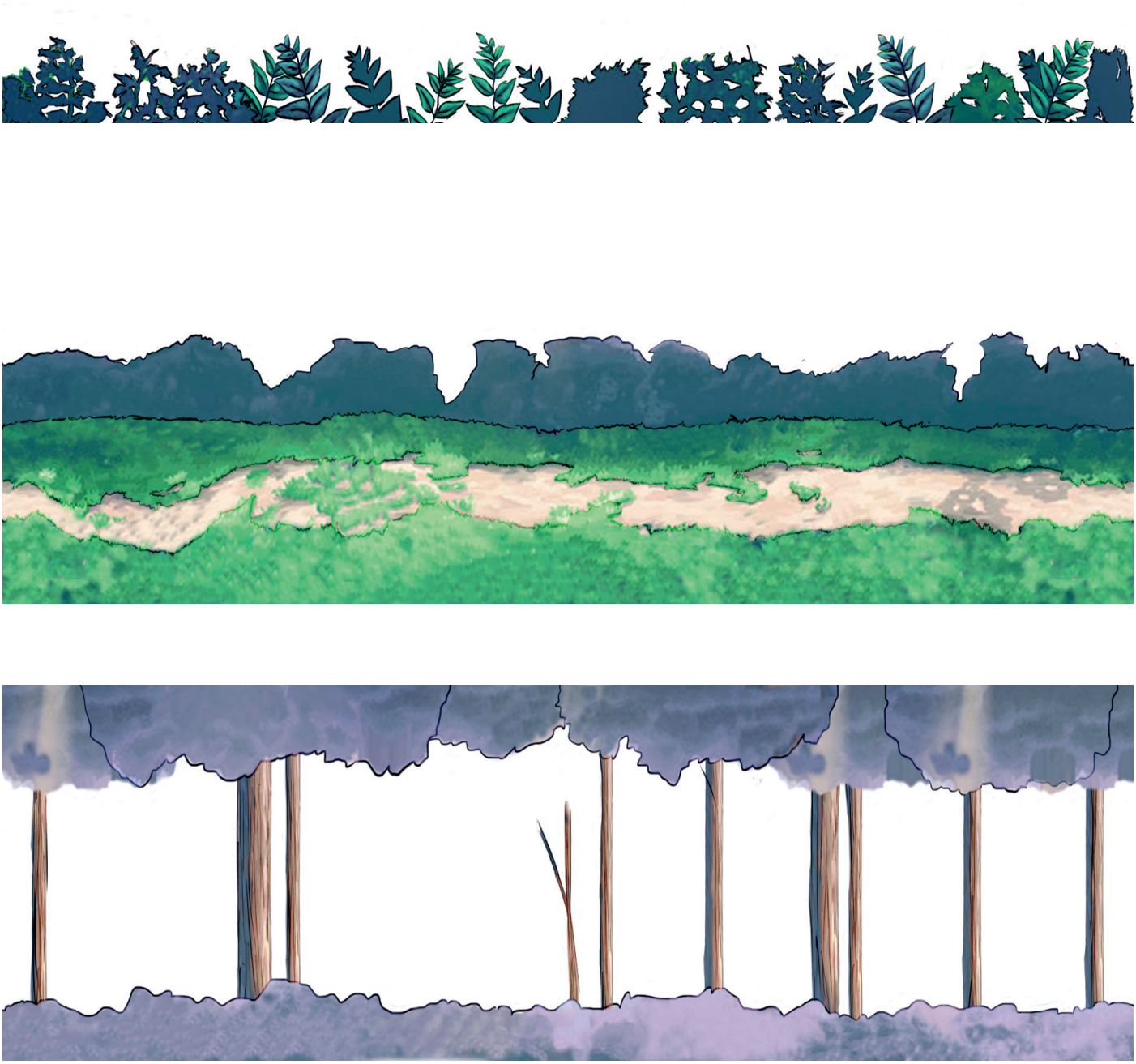
En el nivel 1, la primera escena es un dialogo donde se presenta al héroe de la historia y se explica la misión. Una vez acabado el dialogo pasa automáticamente a la siguiente escena dando comienzo al nivel jugable.

El nivel 2 se compone de también de 2 escenas, la primera una jugable y la segunda otro dialogo.

Y el último nivel se compone también de una escena jugable, esta vez un jefe y una escena de final de juego.

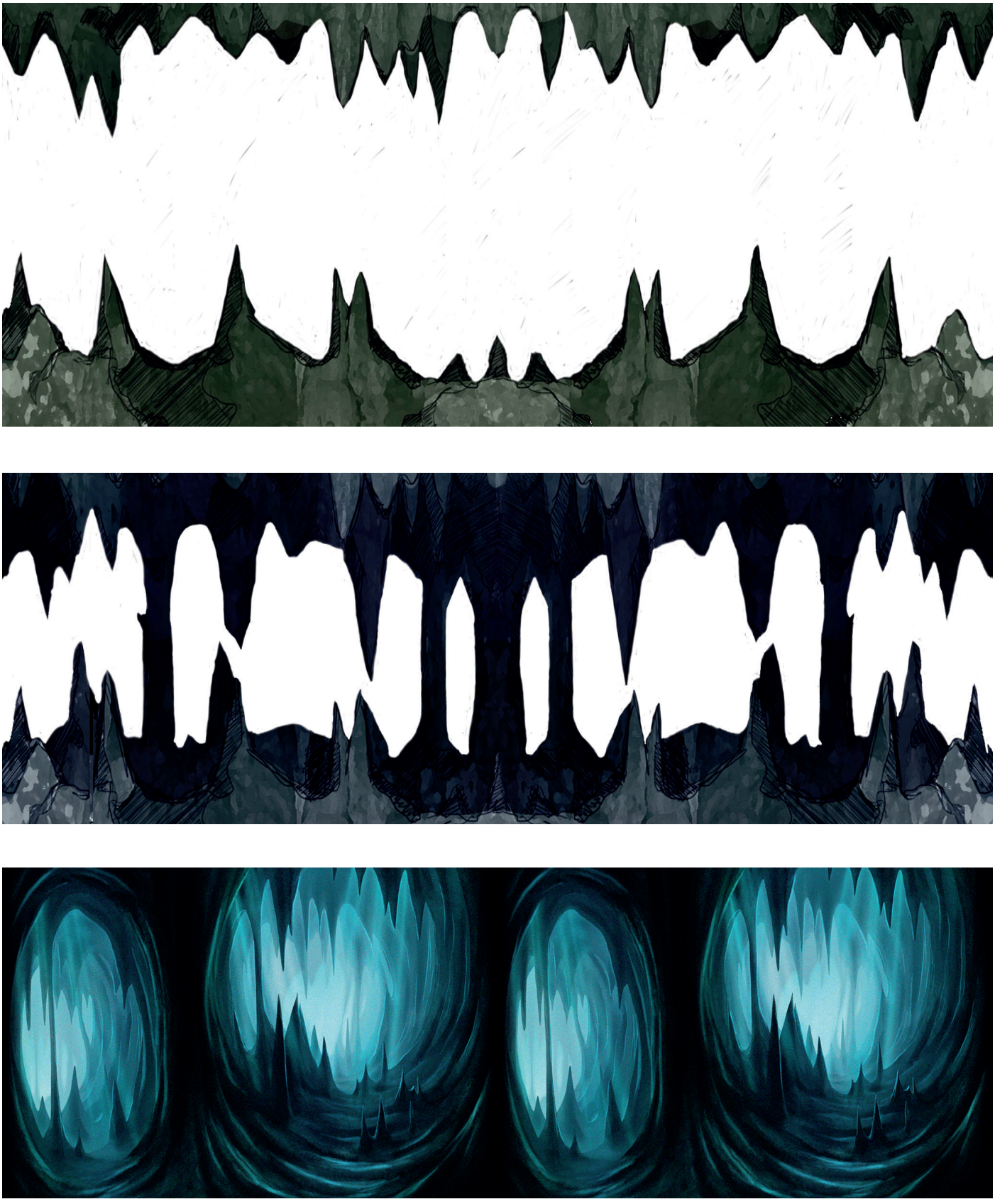
Tanto las escenas de diálogo como jugables tienen varias capas como escenario.

3-Diseño y desarrollo de la aplicación



Escenario bosque

3-Diseño y desarrollo de la aplicación



Escenario cueva

3-Diseño y desarrollo de la aplicación

3.4.3- Jugador

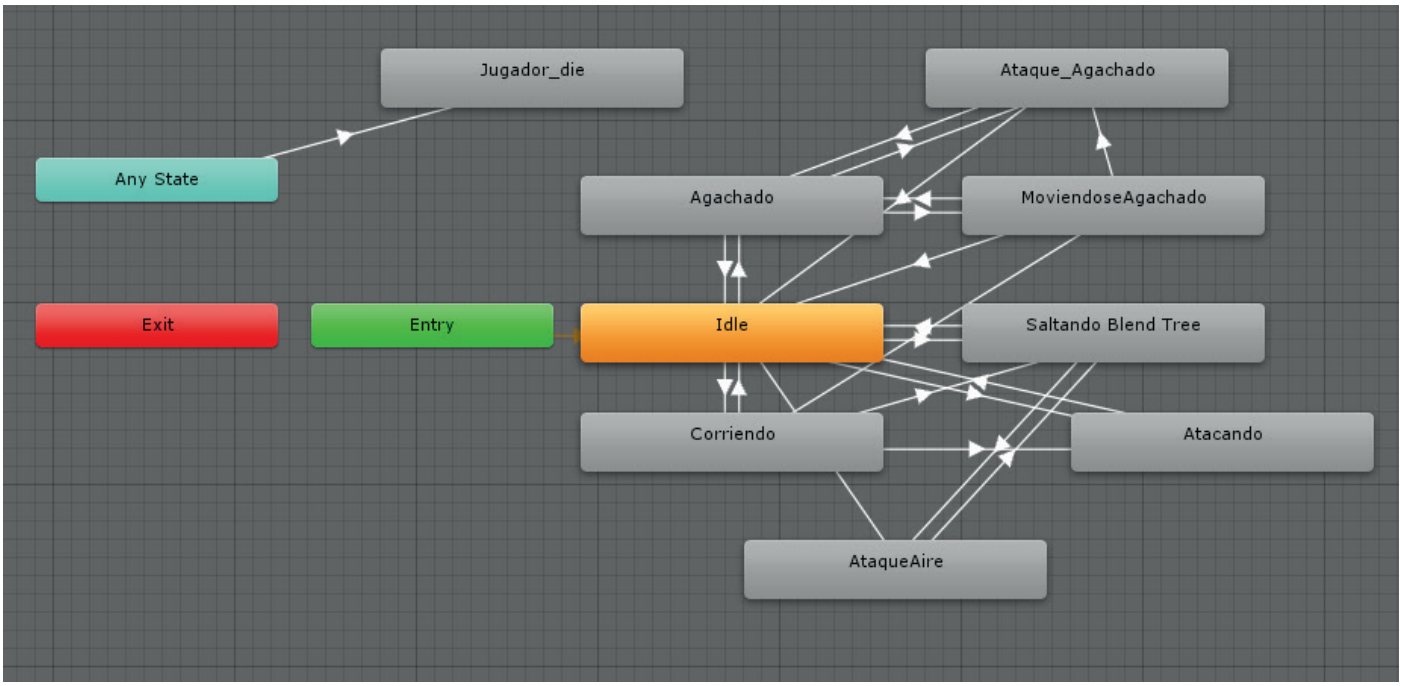
El jugador es el objeto más complejo de todos y es el que mejor debe estar implementado ya que todo el juego gira entorno a él.

Controlaremos a nuestro protagonista con el teclado, las flechas de dirección para moverse y agacharse y los botones z y x para atacar y saltar respectivamente.

El jugador contará con una cantidad de vida que se podrá ver en la parte superior izquierda de la pantalla, el cuál será mostrado por el canvas.

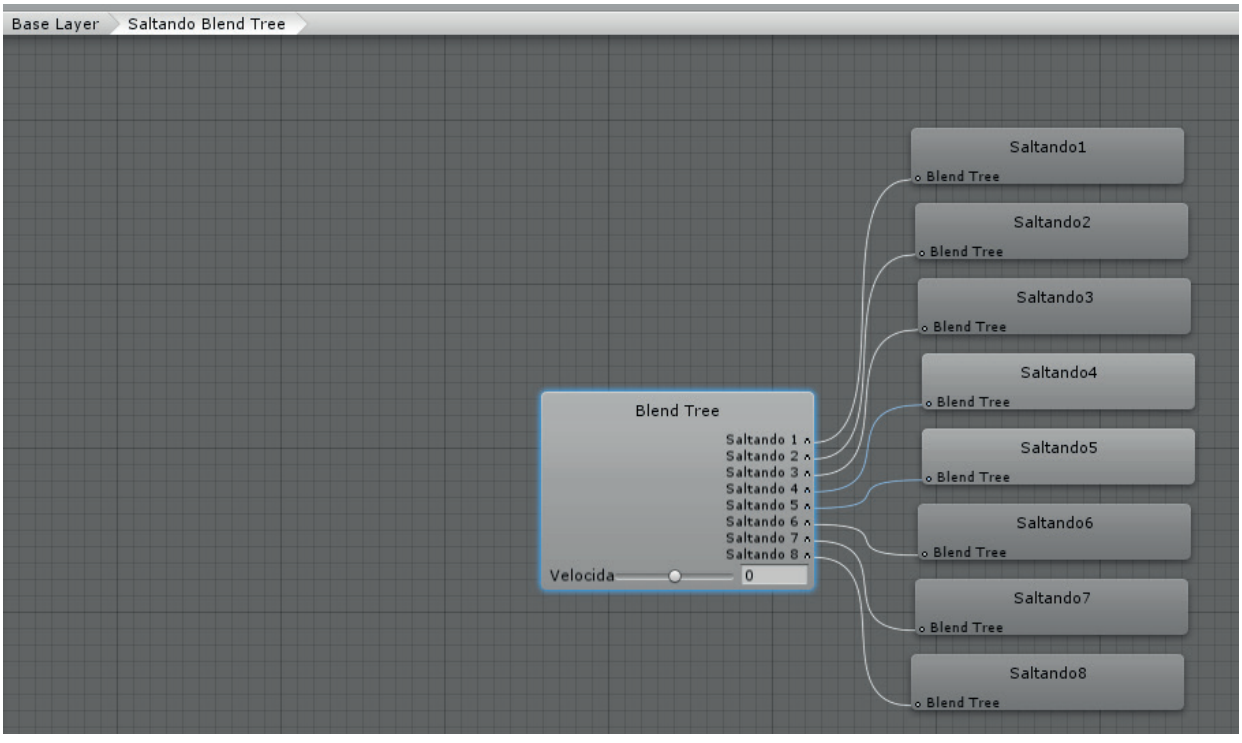
Respecto a los estados que tendrá, son varios como se verá en la siguiente imagen.

3-Diseño y desarrollo de la aplicación



Además de ser el personaje más complejo del juego referente a los estados, también lo es a la hora de programación. Donde tiene varios scripts como el del cálculo de daño, el jugador y el daño.

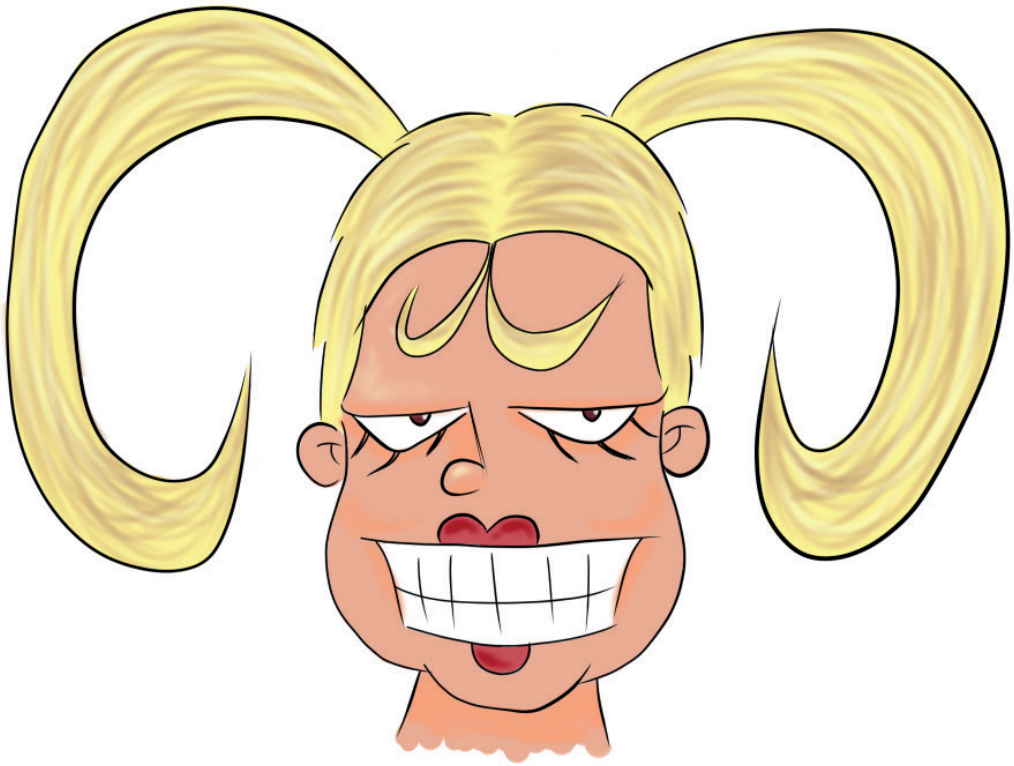
También tiene varios objetos hijos para comprobar si esta en el suelo, si el ataque ha dado al enemigo y un controlador para el daño.



3-Diseño y desarrollo de la aplicación

3.4.4-Personajes secundarios

Los personajes secundarios, son los personajes que solamente salen en la parte del diálogo, para dar más profundidad a la historia y no necesitan ningún tipo de función especifica. Estos en la demostración técnica son el alcalde y su hija.

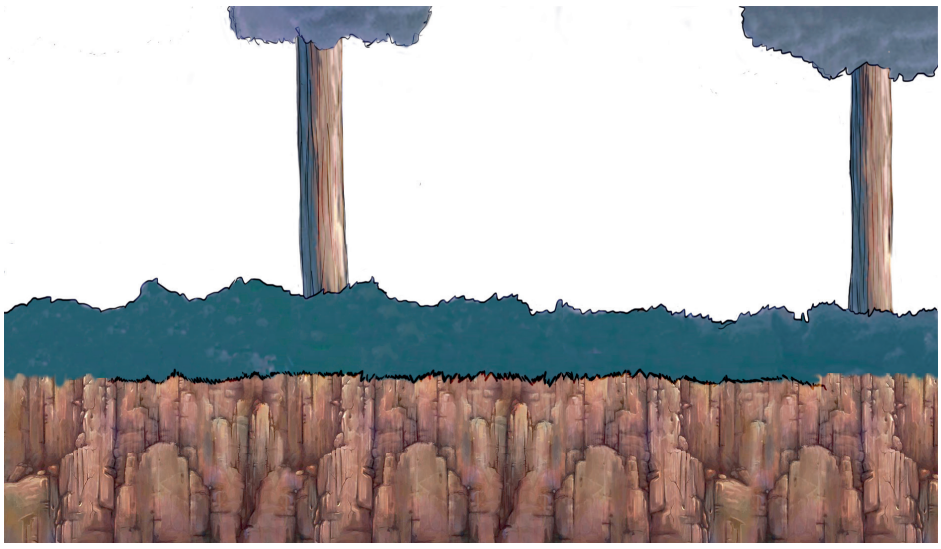


3-Diseño y desarrollo de la aplicación

3.4.5-Enemigos

En esta demostración técnica hay tres tipos de objetos enemigos que causan daño al jugador.

Los primeros son las trampas del escenario que causan daño al jugador al hacer contacto con él. Estos objetos son las picas y los hoyos o espacios vacíos en el escenario.



3-Diseño y desarrollo de la aplicación

Los segundos son los enemigos normales que se dividen en cuatro tipos. Estos enemigos se dividen en tres tipos de orcos y un murciélago.

El primer orco es el orco más básico donde aparte de las funciones de daño y vida que tienen todos los enemigos, es su función de patrullas donde en un principio sólo tiene una patrulla de punto A a punto B, pero con la función creada se pueden colocar los puntos que se quieran para que cada uno tenga una ruta especifica.

3-Diseño y desarrollo de la aplicación

El segundo orco es un arquero donde esta en una posición fija siempre mirando en dirección al jugador y cuando entra dentro de su rango dispara creando un objeto flecha en una posición en especifico con una dirección y una rotación especifica.

3-Diseño y desarrollo de la aplicación

El tercer orco es un lancero el cual también esta en una posición fija y va alterando su vector x para dar la sensación de hacer guardia, el cual cuando se acerca el jugador a cierta distancia este se gira al él y lo embiste hasta que salga fuera de su rango.

3-Diseño y desarrollo de la aplicación

Y el último enemigo de esta clase es el murciélago el cual esta en su zona a que el jugador entre en su rango y lo persigue por todo el escenario hasta que sale de su rango. Una vez esto el enemigo vuelve a su posición inicial.

3-Diseño y desarrollo de la aplicación

Cómo último tipo de enemigo, son los denominados jefes o bosses. En esta demostración técnica solo se ha creado un jefe, el cual es el líder de los orcos. Este como los futuros jefes que se creen en el juego tendrá un mecánica para derrotarlo. La mecánica para este es ir esquivando sus envestidas y esperar a que coja una piedra para atacarlo y que esta le caiga en la cabeza causando daños al boss. Y al repetir esta acción unas veces el boss es derrotado.

3-Diseño y desarrollo de la aplicación

3.4.6-Plataformas

En el transcurso del juego, el jugador se encontrará con varios obstáculos por el camino, alguno de estos obstáculos son las denominadas plataformas.

Habrán varios tipos de plataformas en el juego:

Las estáticas: cómo el nombre indica esta estáticas en el escenario y el jugador no puede traspasar la plataforma.

Las traspasables: son parecidas a las estáticas, pero con la diferencia de que estas puedes traspasarlas tanto de la parte inferior a la superior de la plataforma, como de la superior a la inferior.

Con tiempo de caída: son las plataformas que al tocarlas caen en un tiempo determinado y al cabo de un rato reaparecen.

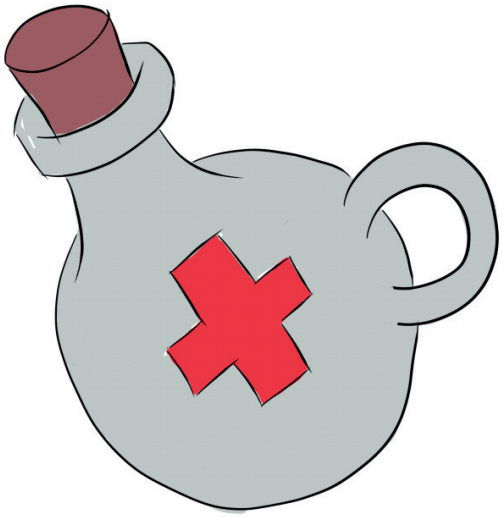
Movibles: son las plataformas que se mueven en el escenario.

3-Diseño y desarrollo de la aplicación

3.4.7-Drops

Al destruir a los enemigos hay un porcentaje de que este suelte un drop al azar. Estos son varias cantidades de dinero o una poción. Los drops están programados para que cuando el jugador los toque estos desaparecen una vez aplicado su efecto.

Los sacos de dinero: Estos sacos aumentan el dinero o la puntuación del jugador, el cual tendrá usos para el menú de compra. La cantidad dependerá de una función random.



Las pociones: Este es el otro objeto que dropea los enemigos y es una poción curativa que regenera un corazón del jugador.

3.4.8-Banda Sonora y efectos sonoros

Aunque puede ser considerado una parte menos importante o destacable del juego. Hasta que se cree una banda sonora propia para el juego, se esta utilizando una banda sonora para configurar las funciones necesarias y poder hacer las pruebas de la demo técnica.

Se pretende crear una banda sonora con el Fl Studio que de una ambientación a un mundo de fantasía y de historias heroicas.

4-Resultados

4.1-Características finales de la aplicación

Finalmente la demo técnica consta de un menú principal, dos diálogos, tres pantallas jugables y un menú final.

El juego empezará con un diálogo para introducir al jugador en la historia. Acto seguido,en las pantallas el jugador irá avanzando en el nivel derrotando a los enemigos que se encuentren en el camino. En cuanto a los enemigos se comportan de la manera explicada anteriormente, cada uno con sus patrones respectivos. Mientras el jugador va avanzando y derrotando a los enemigos, estos soltaran algunos drops que ayudaran al jugador a avanzar.

Una vez llegá al final de la última pantalla se enfrentara al jefe de zona para acabar la misión.

4.2-Descripción de la mecánica y objetivos de la aplicación

El juego se controlará mediante teclado, Las flechas serán las direcciones y agacharse, el botón z para atacar y el botón x para saltar.

Cómo se ha mencionado ante el jugador deberá avanzar derrotando a los enemigos para avanzar en la historia.

4.3-Mercado al que va dirigido y comercialización.

Una vez acabado el juego completamente irá dedicado sobretodo a la gente que le gusta los mundos de fantasía y los juegos 2D. También para aquellos que quieren recordar los juegos de antaño o retro.

Se quiere que el juego tenga varias horas de duración, regujable y sea tanto para hombres como para mujeres.

Para la comercialización, inicialmente se va a subir en Steam Direct (de Steam), en la plataforma de pc y en un futuro se pasará a otras plataformas.

5-Trabajo futuro

El trabajo actual aún esta en desarrollo, por eso hay todavía varias mejoras u objetivos que se tienen que implementar para futuro.

El primer punto es un guión el cual le de más profundidad al juego, junto con los personajes. También habrá muchos más niveles en el futuro, donde habrá niveles con diferentes modos de juego.

Al haber más niveles habrán también varios enemigos, ya sean trampas, enemigos normales y jefes.

En lo referente al lado jugable se añadirá más personajes junto con una selección de estos, donde cada personaje tendrá sus propias características. También se añadirán habilidades y cada héroe tendrá también sus propias habilidades y transformaciones.

Por último crear una banda y efectos sonoros propios para el juego, para poder publicarlo sin problemas.

6-Anexos

6.1-Código

Código Jugador

```
using UnityEngine;
using System.Collections;

public class JugadorV2 : MonoBehaviour {

	[Header (“Movimiento”)]
	[Tooltip (“Velocidad del jugador”)]
	public float velocidadmax = 8f;
	//public float velocidadX = 2f;
	[HideInInspector]
	public bool MirarDerecha = true;
	[Tooltip (“Fuerza del Salto”)]
	public float salto = 7f;
	private float doblesalto = 0f;
	[Tooltip (“Multiplicador de la velocidad del jugador saltando”)]
	public float velocidadSalto = 0.7f;
	[HideInInspector]//se utiliza en el blend tree para el salto
	public float velocidadY;
	[HideInInspector]
	public bool puedeMoverse = true;

	[Header (“Comprobadores del Suelo”)]
	[Tooltip (“Comprobador si esta tocando el Suelo”)]
	public bool estoyenSuelo = false;
	[Tooltip (“Posición del centro del circulo”)]
	public Transform misuelo;
	[Tooltip (“Layer con el cual que se quiere comprobar”)]
	public LayerMask suelo;
	[Tooltip (“Diametro del objeto vacio ComprobadorSuelo”)]
	public float radio =1f;
	[Tooltip (“Comprobador si esta agachado”)]
	public bool estaagachado = false;
	[Tooltip (“Multiplicador de la velocidad del jugador agachado”)]
	public float velocidadAgachado = 0.5f;

	[Header (“Comprobadores del Ataque”)]
	[Tooltip (“Comprobador si esta atacando”)]
	public bool estaatacando = false;
	[Tooltip (“Comprobador si esta atacando agachado”)]
	public bool estaatacandoagachado = false;
	[Tooltip (“Comprobador si esta atacando en el aire”)]
	public bool estaatacandoaire = false;
```

6-Anexos

```
[Tooltip ("Cuenta el tiempo entre ataques")]
public float temporizadorAtaque = 0;
[Tooltip ("Definir el tiempo entre ataques")]
public float tiempoentreAtaques = 0.3f;
[Tooltip ("Colocar el trigger del ataque")]
public Collider2D ataqueTrigger;
public bool estamuerto = false;
//public bool estaherido = false;

Animator misanimaciones;
// Use this for initialization
void Start () {
	misanimaciones = GetComponent<Animator> ();
	ataqueTrigger.enabled = false;

}

// Update is called once per frame
void Update () {


}

void FixedUpdate(){
	/*código comprobación estoy en el suelo*/
	//comprueba si el circulo esta en el layer suelo (del objeto suelo)
	estoyenSuelo = Physics2D.OverlapCircle (misuelo.position,radio,suelo);	//coge los
valores de la posicion misuelo con el radio especifico, chocando con el layer suelo
	misanimaciones.SetBool("EnSuelo", estoyenSuelo);
	misanimaciones.SetFloat ("Velocidady", velocidadY);
	misanimaciones.SetBool ("estaAgachado", estaagachado);
	if (estamuerto == true) {
		misanimaciones.SetBool ("estaMuerto", estamuerto);
		GetComponent<Rigidbody2D> ().velocity = Vector2.zero;
		return;
	}

	//-------------------------------------------------------------------------------------------
-----------------------

	/*código para parar en resbalar*/
	Vector3 VelocityFija = GetComponent<Rigidbody2D> ().velocity;
	VelocityFija.x *= 0.75f;
	/* código mover jugador */
	float moverse = Input.GetAxis ("Horizontal");
```

6-Anexos

```
		if (puedeMoverse) {
			GetComponent<Rigidbody2D> ().velocity = new Vector2 (velocidadAgachado
* velocidadSalto * moverse * velocidadmax, GetComponent<Rigidbody2D> ().velocity.y);
			velocidadY = GetComponent<Rigidbody2D> ().velocity.y;
		} else {
			GetComponent<Rigidbody2D> ().velocity = new Vector2 (0 * velocidadAga-
chado * velocidadSalto * moverse * velocidadmax, GetComponent<Rigidbody2D> ().velocity.y);
			velocidadY = GetComponent<Rigidbody2D> ().velocity.y;
		}
		//-----------------------------------------------------------------------------------------
--------------------
		/*Girar Sprite*/
		if (moverse > 0 && !MirarDerecha)
			GirarSprite ();
		else {
			if (moverse < 0 && MirarDerecha)
				GirarSprite ();
		}

		//-----------------------------------------------------------------------------------------
-----------------------
		/*código agacharse*/
		if (estaagachado) {
			velocidadAgachado = 0.5f;
		} else {
			velocidadAgachado = 1f;
		}

		if (Input.GetKey (KeyCode.DownArrow) && estoyenSuelo) {
			estaagachado = true;
		} else {
			estaagachado = false;
		}
		//-----------------------------------------------------------------------------------------
-----------------------
		/*código salto*/
	if (estaatacando == false && estaatacandoagachado == false && estaatacandoaire==false) {
		if (estoyenSuelo) {
			velocidadSalto = 1f;
			if (!Input.GetKey (KeyCode.X)) {
				doblesalto = 0;
			}
		} else {
			velocidadSalto = 0.7f;
```

6-Anexos

```
		}
		if (Input.GetKey (KeyCode.X) && (estoyenSuelo || doblesalto == 2)) {
			//código salto con addforce
			GetComponent<Rigidbody2D> ().velocity = new Vector2 (GetComponent<Rigidbody2D> ().velocity.x, 0f); //-->para evitar sumar fuerzas
			GetComponent<Rigidbody2D> ().AddForce (new Vector2 (0, salto));
			//código salto con velocity
			//GetComponent<Rigidbody2D> ().velocity = new Vector2 (GetComponent<Rigidbody2D> ().velocity.x, salto);
			if (doblesalto == 0 || doblesalto == 2) {
				doblesalto++;
			}

		}
		if (Input.GetKey (KeyCode.X) && !estoyenSuelo && doblesalto == 0) { //tambien podra ser keycode.uparrow
			//código salto con addforce
			GetComponent<Rigidbody2D> ().velocity = new Vector2 (GetComponent<Rigidbody2D> ().velocity.x, 0f); //-->para evitar sumar fuerzas
			GetComponent<Rigidbody2D> ().AddForce (new Vector2 (0, salto));
			//código salto con velocity
			//GetComponent<Rigidbody2D> ().velocity = new Vector2 (GetComponent<Rigidbody2D> ().velocity.x, salto);
			doblesalto = doblesalto + 2;

		}
		if (!Input.GetKey (KeyCode.X) && doblesalto == 1) {
			doblesalto = 2;
		}
	}
	//-------------------------------------------------------------------------------------------------------------
	/*codigo atacar*/
	if (Input.GetKey (KeyCode.Z) && (!estaatacando || !estaatacandoagachado || !estaatacandoaire) && puedeMoverse) {
		if (estoyenSuelo && !estaagachado) {
			puedeMoverse = false;
			Atacar ();
		}
		if (estaagachado) {
			puedeMoverse = false;
			AtacarAgachado ();
		}
		if (!estoyenSuelo) {
			puedeMoverse = false;
```

6-Anexos

```
                AtacarAire ();
            }

        }

        if (estaatacando || estaatacandoagachado || estaatacandoaire) {
            if (temporizadorAtaque > 0) {
                temporizadorAtaque -= Time.deltaTime;
            } else {
                if (estaatacando) {
                    estaatacando = false;
                    ataqueTrigger.enabled = false;
                }
                if (estaatacandoagachado) {
                    estaatacandoagachado = false;
                    ataqueTrigger.enabled = false;
                }
                if (estaatacandoaire) {
                    estaatacandoaire = false;
                    ataqueTrigger.enabled = false;
                }
                puedeMoverse = true;
            }
            //problemas a causa de un if
            misanimaciones.SetBool (“estaAtacando”, estaatacando);
            misanimaciones.SetBool (“estaAtacandoAgachado”, estaatacandoagachado);
            misanimaciones.SetBool (“estaAtacandoAire”, estaatacandoaire);



        }

        //-------------------------------------------------------------------------------------------------
---------------------
        /*código de animaciones*/
        //añadir la animacion absoluto del movimiento
        misanimaciones.SetFloat (“velocidad”, Mathf.Abs (GetComponent<Rigidbody2D> ().veloci-
ty.x));
}

void GirarSprite(){
        MirarDerecha = !MirarDerecha;
        Vector3 cambio = transform.localScale;  //Se coge en la variable “cambio” los valores del
localScale -->localScale == la escala del transform
        cambio.x *= -1;                 //Se invierte del valor x del localScale
```

6-Anexos

```
        transform.localScale = cambio;

}

void Atacar(){
        estaatacando = true;
        temporizadorAtaque = tiempoentreAtaques;
        ataqueTrigger.enabled = true;

}
void AtacarAgachado(){
        estaatacandoagachado = true;
        temporizadorAtaque = tiempoentreAtaques;
        ataqueTrigger.enabled = true;

}

void AtacarAire(){
        estaatacandoaire = true;
        temporizadorAtaque = tiempoentreAtaques;
        ataqueTrigger.enabled = true;

}

}
```

Código cálculo de daño

```
using UnityEngine;
using System.Collections;

public class CalculoDanyoJugador : MonoBehaviour {

        [Header ("Comprobadores de vida")]
        //[Tooltip ("Gameobject de la vida")]
        //public Text Textovida;
        [Tooltip ("Vida actual del jugador")]
        public int vida;
        private int vidamaxima = 5;
        [Tooltip ("Distancia que se movera al ser golpeado")]//explicar bien cada caso
        public float retraso;
        [Tooltip ("Tiempo entre retrasos")]
        public float retrasoLongitud;
        [Tooltip ("Contador del retraso")]
        public float retrasoContador;
```

6-Anexos

```
	[Tooltip (“Retraso de por la derecha”)]
	public bool retrasoDerecha;

	//corrutina invencible
	[Tooltip (“Render del objeto invencible”)]
	public Renderer rendinvencible;

	public GameObject prota;


	// Use this for initialization
	void Start () {
		vida = vidamaxima;
		rendinvencible.GetComponent<Renderer>();
	}

	// Update is called once per frame
	void Update () {

	}

	/*Calculo de daño y modificar vida*/
	void OnTriggerEnter2D(Collider2D obj){

		if ((obj.tag == “Enemigo”|| obj.tag ==”AtqEnemDist”) && !GestorJuego.controlador.
GetInvulnerable() /*&& ataqueTrigger.enabled==false*/) {
			//daño del objeto con la cantidad puesta en la script

			vida-=obj.GetComponent<Danyo>().danyo;

			if (obj.tag == “AtqEnemDist”) {
				Destroy (obj.gameObject);//asi destruye el objeto(ataque enemigo dis-
tancia)
			}
			if (vida <= 0) {
				vida = 0;
				Debug.Log(“Estas Muerto”);
				prota.GetComponent<JugadorV2> ().estamuerto = true;

			}
			StartCoroutine(Invulnerabilidad());

		}

	}

	void OnTriggerStay2D(Collider2D obj){
```

6-Anexos

```
        if ((obj.tag == “EnemigoZona”) && !GestorJuego.controlador.GetInvulnerable ()) {
            //daño del objeto con la cantidad puesta en la script

            vida -= obj.GetComponent<Danyo> ().danyo;

            if (vida <= 0) {
                vida = 0;

                Debug.Log (“Estas Muerto”);
                prota.GetComponent<JugadorV2> ().estamuerto = true;


            }

            StartCoroutine (Invulnerabilidad ());

        }
    }



    //Corrutina invencible
    IEnumerator Invulnerabilidad(){
        yield return new WaitForSeconds (0.001f);//para que la nave enemiga sufra el golpe
        GestorJuego.controlador.SetInvulnerable(true);


        bool final = false;
        double tiempo = 0;
        do {
            rendinvencible.enabled = !rendinvencible.enabled;//parpadeo
            yield return new WaitForSeconds (0.1f);
            tiempo += 0.1;
            if (tiempo >= 1.5) {
                final = true;
            }
        } while(!final);

        rendinvencible.enabled = true;
        GestorJuego.controlador.SetInvulnerable (false);


    }
}
```

6-Anexos

6.2-Costes

CONCEPTO	EUROS/MES	EUROS/AÑO	INVERSIÓN	VIDA ÚTIL	AMORTIZAR
Luz	70’00	840’00			
Agua	20’00	240’00			
Internet	25’95	311’40			
Hardware			879’00	4 años	219’75
Software		735,92 (Adobe CC + Unity PRO)	340’47 (SAI + Spine)	4 años	85’12
Cuota de Autónomos	300’00 801’12				
Material	40’00	480’00			
Periféricos			129’95	4 años	32’49
Salario	800’00	9600’00			
Total anual		14657,86		Coste / hora	8,14
Horas de trabajo		1800 h.			

Horas x Coste/hora = Coste del proyecto (actualmente).
560 h. x 8,14 = 4558,4 Euros

Horas dedicadas al proyecto.
4 h./dia x 5 dias/semana x 7meses; 20h./semana x 4semanas/mes = 80h/mes x 7meses = 560h.

Actualmente distribución y ventas esta parado esperando a llegar a fase beta, pero se tiene planteado usar plataformas digitales de videojuegos como por ejemplo Steam para su distribución.

La promoción estaría a cargo de nosotros como estudio, entre redes sociales y foros, pero como comente anteriormente este tema esta parado ya que nos centramos en elaborar el juego.

Antes de presentar la propuesta en crowdfunding registraríamos la obra para conservar nuestra propiedad intelectual.

7-Bibliografía

https://unity3d.com/es

https://www.youtube.com/

https://steamcommunity.com/games/593110/announcements/detail/1328973169870947116

http://www.hobbyconsolas.com

http://www.3djuegos.com/

http://www.randalsmonday.com/es/

http://deusexmachina.es/entrevista-a-juanan-y-toni-pascual-creadores-de-randals-monday/

http://meristation.as.com/

http://es.esotericsoftware.com/

https://www.systemax.jp/en/sai/

http://www.adobe.com/es/products/photoshop.html