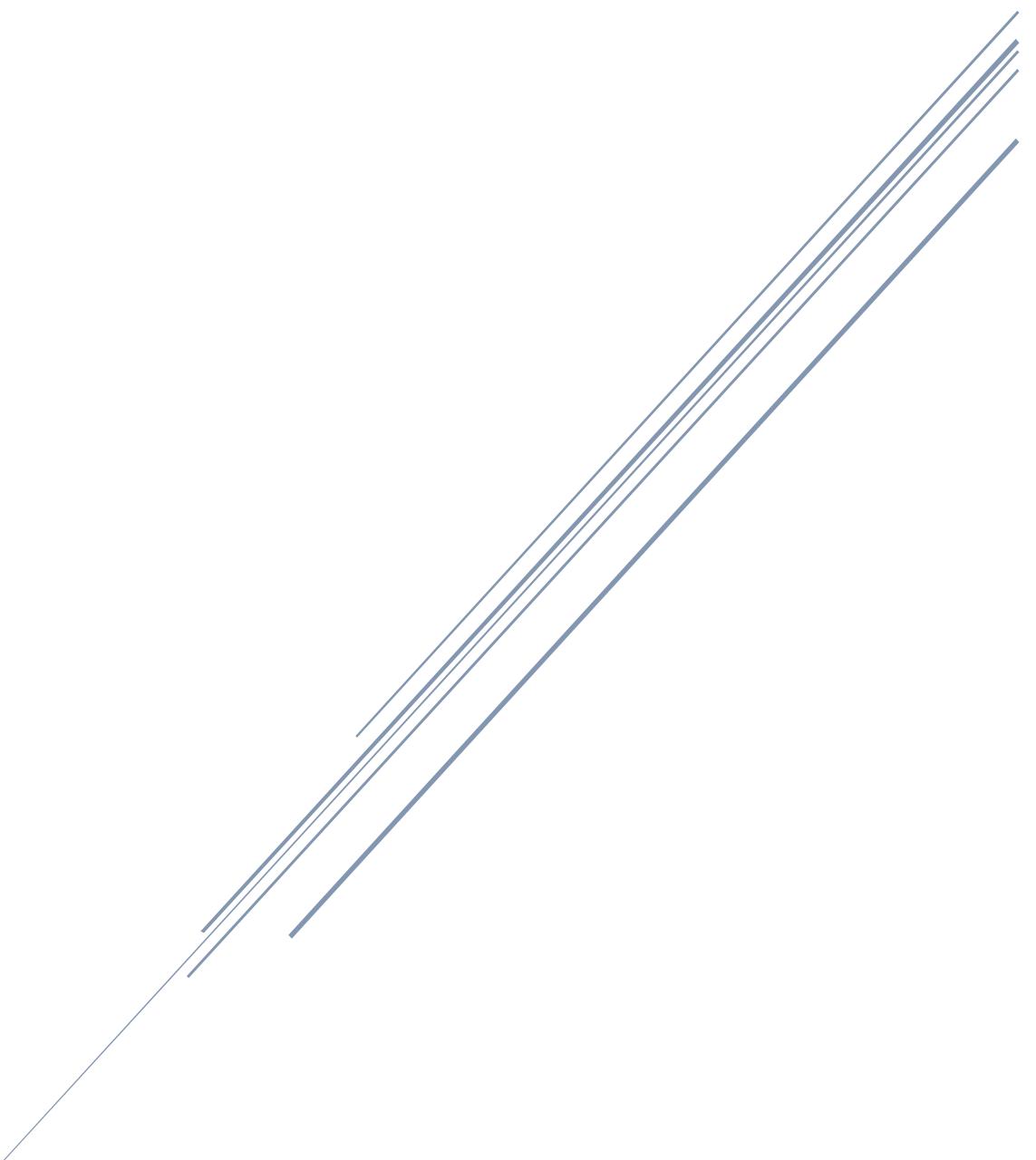


PROYECTO FINAL DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA

INSTITUTO TECNOLÓGICO EDIX



PABLO HITOS MADRID
76419088-R 2ºDAM

2. Introducción, justificación y objetivos: se describirán la motivación que ha originado la realización del TFG, así como de una breve descripción de los objetivos generales que se quieren alcanzar con el trabajo presentado.

Introducción:

El trabajo realizado como proyecto final de grado se ha centrado en la creación de un videojuego desde 0 con el motor gráfico de Unity. Uno de los motores más famosos en la creación de videojuegos y además con posibilidad de usarse gratuitamente.

Desde primera hora se ideó hacer un juego especialmente en 2D para que llegase al mayor número de plataformas posibles, es decir: tanto equipos de ordenador de gama baja y alta y a smartphones de gama alta y baja.

El juego en cuestión se llama Spriters Island, es un juego de plataformas en scroll 2D con estilo pixel art ya que intenta imitar los estilos de diseño más tradicionales. Este juego bebe directamente de los juegos más famosos del mercado y aquellos que forman parte de la “Old School” como : Super Mario Bros, Donkey Kong Country u otros más modernos como Super Meat Boy.

El juego a pesar de ser un plataformas, desde su inicio y planificación se mantuvo con una fuerte premisa de que no tendría un carácter monótono en el que solo sea eliminar enemigos y llegar al final de la pantalla.

Se procuró que dependiendo de la pantalla estas tengan objetivos distintos para poder superarlas, como coger un diamante, coger un número de frutas o coger un número de frutas en el nivel mientras se encuentran limitados por un “timer”.

Durante su planificación, se pensó que sería muy interesante elegir con qué personaje el user quería superar el nivel, por lo que el juego dispone de un selector de personajes con total de 7 de ellos. Cada uno de los personajes tiene un diseño muy distinto del resto.

Justificación:

El motivo por el que se decidió hacer como proyecto la creación de un videojuego es claro y simple.

A día de hoy y gracias a las nuevas tecnologías el mundo del videojuego ha desarrollado un crecimiento de unos niveles increíblemente notables tanto económico como en popularidad. No es necesario hacer un estudio exhaustivo para comprobar las grandes inversiones que se están haciendo en este sector, ya que hay videojuegos que fácilmente se encuentran al nivel de una super producción de Hollywood, y no solo eso, si no que actores de gran calibre y renombre prestan su imagen y sus voces para ellos además del nivel musical que muchos de ellos presentan como sus bandas sonoras (Cyber Punk 2077, Death Stranding, Metal Gear Solid) .

Se ha querido centrar en la creación de un videojuego porque dicho sector a día de hoy es considerado como un músculo financiero equiparable al mundo de la música o del cine, por ello, generaba bastante curiosidad introducirse aunque sea mínimamente en el sector y seguir aprendiendo de este por si algún día existía la posibilidad de profesionalizarse en el sector.

El avance de las nuevas tecnologías, hacen que el mundo del streaming, los youtubers e influencers se encuentren mas que nunca en la vanguardia, esto ha supuesto un golpe de suerte para el mundo de los videojuegos, ya que hace años era un sector claramente denostado y asociado al mundo de la adicción o algo a evitar.

Como se iba exponiendo, la aparición de los Gameplays en plataformas de video ha hecho que el sector del videojuego tenga mas relevancia y los streamers, influencers y youtubers son claramente conscientes de ello y aprovechan el mundo del videojuego para proporcionar horas y horas de entretenimiento a sus seguidores.

El desarrollo de motores como Unity, GameMaker, etc... ha hecho que el acceso al desarrollo del mundo del videojuego sea mas simple. Esto junto con el gran consumo de Twitch por parte de los usuarios propició que el número de desarrolladores de videojuegos indies crezca muchísimo.

Por todo lo expuesto, se quiso aprovechar la gran fama que ha ido adquiriendo motores como Unity y hacerlo el eje del proyecto, gracias a este ascenso, el acceso a la información para crear proyectos (ya sea por youtube, stackoverflow, foros de unity) es abismal y se ha multiplicado, por lo que era una reseñable baza y gran oportunidad para poder intentar crear algo llamativo y que cumpliera las expectativas como proyecto.

3. Agradecimientos

Quiero agradecer a las siguientes personas este proyecto realizado porque considero que sin la ayuda de ellas todo habría sido mas complicado:

A toda mi familia: Muchas gracias a todos por el apoyo mostrado y especialmente a mi madre por apostar por mí y por el apoyo económico que ha supuesto realizar estos estudios.

A uno de mis mejores amigos David Zafra: Muchas gracias David por estar ahí desde el principio ayudándome con los estudios, gracias por tu infinita paciencia mientras aprendía a programar.

A mis amigos Pedro Romo y Christian Robles: Muchas gracias Pedro por tus consejos, tu ayuda y tu sabiduría. Muchas gracias Christian por tu betatesting y por tu paciencia a la hora de probar mi proyecto.

A Mary Fernández: Gracias.

A todos los profesores y miembros del Instituto Tecnológico Edix: Sin vuestra ayuda y vuestros consejos esto no podía estar sucediendo.

A todo el equipo del motor Unity.

A Pixel Frog, el creador de diferentes Assets como Pixel Adventure 1 y 2 los cuales se usaron en este proyecto. Este contenido me ha sido muy útil.

A Ansimuz, el creador de los 3 Assets de Sunny Land (Sunny Land, Sunny Land Forest, Sunny Land Wood). Los assets de este creador son una base fundamental de este proyecto.

A todos los creadores de assets en itch.io, a todos los compositores de la música usada en este juego y a muchos más.

4. Palabras clave, una tabla con acrónimos o explicación de determinados conceptos en caso de ser necesarios.

Animator Controller: es una máquina de estados que permite organizar y mantener un conjunto de animaciones en un objeto. Normalmente un objeto tiene varias animaciones que cambian dependiendo del estado del objeto (parado, corriendo, saltando...) y gracias al Animator Controller se puede hacer fácilmente.

Asset: Un asset es una representación de cualquier ítem que puede ser utilizado en el juego o proyecto. Un asset podría venir de un archivo creado fuera de Unity, tal como un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos que Unity soporta. También hay otros tipos de asset que pueden ser creados dentro de Unity, tal como un Animator Controller, un Audio Mixer o una Render Texture.

AI: Inteligencia artificial.

Collider: Definen la forma de un objeto para los propósitos de colisiones físicas. Un collider, el cual es invisible, necesita estar con la misma forma exacta que el mesh del objeto y de hecho, una aproximación a menudo es más eficiente e indistinguible en el juego.

Event Trigger: El Event Trigger (Activar/desactivar eventos) recibe eventos del Event System (Sistema de eventos) y llama las funciones registradas para cada evento.

El Event Trigger puede ser utilizado para especificar funciones que se quieren llamar para cada evento del sistema de eventos. Se pueden asignar varias funciones a un solo evento y cuando el Event Trigger recibe ese evento, este llamará a todas esas funciones.

GameObject: Son objetos fundamentales en Unity que representan personajes, props, y el escenario. Estos no logran nada por sí mismos pero funcionan como contenedoras para Components, que implementan la verdadera funcionalidad.

Physic Material: Es utilizado para ajustar la fricción y efectos de rebote de objetos que chocan.

Prefabs: son objetos reutilizables y creados con una serie de características dentro de la vista proyecto, que serán instanciados en el videojuego cada vez que se estime oportuno y tantas veces como sea necesario. Es decir, permiten crear ‘copias’ fácilmente con una serie de ventajas.

Rigidbody: Es aquello que permite a sus **GameObjects** actuar bajo el control de la física. El Rigidbody puede recibir fuerza y torque para hacer que sus objetos se muevan en una manera realista. Cualquier GameObject debe contener un Rigidbody para ser influenciado por gravedad.

Script: Basicamente es el código.

Sprite Renderer Es aquello que permite mostrar imágenes como **Sprites** para su uso en escenas 2D y 3D.

5.Índice

ÍNDICE:

Portada	PAG 0
2. Introducción, justificación y objetivos:	PAG 1
3. Agradecimientos	PAG 3
4. Palabras clave	PAG 4
6. Resumen	PAG 7
7. Módulos formativos aplicados en el trabajo	PAG 9
8. Herramientas/Lenguajes utilizados	PAG 10
9. Componentes del equipo y aportación	PAG 12
10. Fases del proyecto	PAG 12
11. Conclusiones y mejoras del proyecto	PAG 88
12. Bibliografía	PAG 88
13. Anexos	PAG 89

6. Resumen: exposición clara y sucinta del tema tratado en el trabajo en todas sus partes .

Descripción y alcance del proyecto:

Como se indicó en puntos anteriores el juego se llama Spriters Island, es un juego creado para plataformas como PC, Android y navegador web.

Dicho juego está dentro de los conocidos como plataformas siguiendo una línea de juego clásico plataformero en Scroll 2D horizontal que bebe directamente de grandes juegos como Super Mario Bros, Donkey Kong u otros similares como Super Meat Boy.

El juego no está basado en aquellos típicos de llegar al final de la pantalla y listo, sino que cada nivel tiene unos objetivos diferentes que es necesario cumplirse para poder pasar de pantalla.

Por ejemplo: En el nivel 1 es necesario encontrar un diamante para poder pasar de nivel, en el nivel 2 es necesario coger todas las frutas que se encuentran en todo el nivel para poder pasarlo y finalmente en el tercero es necesario coger todas las frutas del nivel pero antes de que el tiempo se acabe.

Los niveles del juego son desbloqueables, es decir conforme se avance de pantalla se podrán ir jugando nuevos niveles, se enfatiza en este hecho ya que el juego dispone de un selector de niveles por lo que si no se pasa el nivel 1, el nivel 2 no desbloqueará y así sucesivamente.

Dentro de esa pantalla de selección se cuenta con un selector de personajes, por lo que se podrá elegir entre 7 personajes diferentes para poder ir completando las pantallas.

El juego consta de unos 10 enemigos que total, es decir, hay enemigos terrestres como setas, plantas que lanzan balas o cerdos que cambian su estado si reciben daño, también hay enemigos voladores como pájaros azules con diferentes trayectorias de vuelo, murciélagos que se encuentran en cavernas, dragones voladores que son más rápido que

el resto de enemigos voladores, abejas que lanzan aguijones u otro tipo de abejas que una vez detectada tu presencia te lanzan el aguijón, por último también hay un fantasma que te persigue a lo largo de toda la pantalla.

Aparte de los enemigos también hay trampas como bolas de pinchos que giran a diferentes grados, bloques de pinchos que suben y bajan, pinchos, lava o precipicios.

El player tiene un total de 4 vidas que puede perder a lo largo de la pantalla, en caso de perderlas el nivel se reiniciará todas las veces que esto ocurra.

En cada nivel hay monedas de diferente valor que el player irá recogiendo y harán que sumen puntos a su score, aparte el score podrá ir subiendo cuanto mas enemigos elimine saltando encima de las cabezas de estos.

El juego dispone de todos accesorios tales como trampolines o power ups en forma de gemas que pueden aumentar durante 10 segundos la velocidad del player o el salto de este, una vez transcurridos los 10 segundos el player volverá a tener sus valores normales.

Los tres niveles disponen de temáticas diferentes y una vez completados saldrá una pantalla de agradecimiento y juego completado.

Alcance del proyecto:

En cuanto al alcance del proyecto se indicó, este juego va destinado a PC, Android y Navegador Web. Los requisitos de cada dispositivo se detallan en el GDD y en el manual de ayuda que se creó expresamente, aunque en resumen, se puede decir que este juego fue diseñado en 2D con la intención de que sea ejecutado en todos los dispositivos posibles, ya sean de gama alta o gama baja.

Para que el juego llegué al mayor número de usuarios posibles se subió a plataformas como Simmer.io e Itch.io y aparte se podrá jugar en la página web del propio proyecto.

Este juego va destinado a un público joven a gente mayor de entre unos 8 años a 65 años.

A parte de todo esto para darle mayor publicidad se han subido gameplays del juego a youtube para que puedan verlo el mayor número de usuarios.

7. Módulos formativos aplicados en el trabajo

Inglés: Se ha usado la asignatura de inglés en este proyecto ya que todo se encuentra en dicho idioma.

El juego se encuentra totalmente en inglés, es decir, sus menús, mensajes etc.. Aparte los comentarios del código se encuentran totalmente en inglés también además de la nomenclatura de dichos códigos.

La página web del proyecto también se encuentra en inglés en su totalidad.

Finalmente se hizo un manual de ayuda e información que se encuentra totalmente explicado en inglés.

Programación: Dado que existen numerosos Scripts que hacen funcionar el juego. Esta es una de las asignaturas que mayormente se ha puesto en práctica. El lenguaje que se ha utilizado es el de C# o C Sharp. El proyecto contiene un total de 30 Scripts con funciones muy concretas cada uno.

Lenguaje de Marcas: Se ha creado una página web que recoge información de la empresa e información del videojuego. Esta página web está creada en Html5 y CSS3. En dicha página existe la posibilidad de poder jugar incluso al juego.

Programación Multimedia y Dispositivos Móviles: El eje del proyecto está basado en Unity por lo que esta asignatura es la que mas se ha usado en el proyecto. No solo se ha usado esta asignatura si no que gracias al proyecto se ha tenido la oportunidad de ahondar mucho más en el funcionamiento de Unity y sus características.

Finalmente, en vez de usar Android Studio se ha usado BlueStacks 5 que es un emulador de Android muy recomendado también y especializado en el tema de apps de videojuegos.

De hecho no sufre ralentizaciones ni problemas de ningún tipo ya que Android Studio consumía bastante mas e iba mas lento.

Desarrollo de interfaces: Se ha creado un manual para el usuario completamente en inglés donde se explica todo, por ejemplo: método de instalación, requisitos mínimos, información del juego, niveles, tipos de enemigos, etc... todo ello acompañado de imágenes ilustrativas además.

También se ha creado un autoinstalador en su versión para PC.

8. Herramientas/Lenguajes utilizados:

Unity Engine: Este es el pilar del proyecto, este motor gratuito es el que ha permitido usar crear todo el videojuego desde el principio hasta el final. La versión utilizada para crear el proyecto para Windows, Android y Web GL es: Unity 2020.3.15f1 (64-bit).

Dentro de Unity se han instalado distintos Assets que permitirá desarrollar nuestro proyecto, tales como: **TextMesh Pro** (nos permitirá hacer los textos más vistosos). Distintos Assets como Pixel Adventure y Sunny Land, entre otros muchos.

Para poder crear los Scripts del videojuego en Unity se ha utilizado el lenguaje de programación **C Sharp o C#**, este ha sido una pieza vital.

Para poder hacer los Scripts vamos a instalar Visual Studio de Microsoft. Este programa será acompañado de ciertos plugins que nos ayudará en nuestra tarea (autocompletados, palabras en color, etc....).

Programas y herramientas para la creación del juego.

Visual Studio 2019: Se ha usado este *IDE para Windows el cual* es compatible con múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, etc....

Esta herramienta ha sido vital para poder crear los Scripts ya que esta iba en guiando en caso de fallos e informaba exactamente si había algún error en la sintaxis usada.

A parte de esta herramienta se instalaron una serie de plugins que ayudaron y facilitaron mucho mas la tarea de programación. Se instalaron plugins de autocompletado, plugins que guiaban aún mas a la hora de programar, etc....

Piskel: Este programa ha permitido editar sprites ya sean de la Store de Unity o de alguna pagina web especializada en Sprites. Uno de los enemigos especialmente ha sido trabajado con Piskel. El enemigo en concreto ha sido “Zombee”, una variante de una abeja descargada en el Asset Store.

Google Translate y DeepL: Cuando se ha necesitado consultar algo en inglés o enfocar algún comentario en los scripts.

Help N Doc: Es una herramienta de creación de documentación de ayuda con la que se ha creado el manual del videojuego en inglés.

Ino Setup: Para crear el autoinstalador.

BlueStacks5: Emulador de Android que ha permitido ejecutar el juego para ver si funcionaba.

Itch.io y Simmer.io: Plataformas Online donde se ha podido colgar el juego para jugar desde el navegador.

Youtube: Plataforma de videos donde se colgó un gameplay del mismo.

Lenguajes y herramientas para la creación de la Página Web:

Visual Studio Code: [Editor de código fuente](#) que se ha usado en Windows junto con una serie de plugins que han ayudado a la edición relativa de la página web.

Html5: es el estándar que se ha usado para definir la estructura y el contenido de una página Web.

CSS3: Es lo que ha permitido generar el diseño visual de páginas web e interfaces de usuario.

9. Componentes del equipo y aportación realizada por cada estudiante (en caso de no ser un trabajo individual)

El trabajo se ha realizado de manera individual por lo que todo el proceso ha sido a cargo de una sola persona: Creación del videojuego, creación de la página web, creación del manual de instrucciones, autoinstalador, etc....

10. Fases del proyecto, siendo la parte con el grueso del trabajo. Entre otros, podría constar de los siguientes puntos.

Analisis: Como ya se indicó en el GDD y en otros puntos del proyecto el eje central del mismo se basa en Unity, concretamente se ha usado la versión Unity 2020.3.15f1 (64-bit).

Se descargó de su página oficial

<https://unity3d.com/es/get-unity/download>

Descargar Unity

Bienvenido! Está aquí porque desea descargar Unity, la plataforma de desarrollo más popular del mundo para crear juegos multiplataforma y experiencias interactivas 2D y 3D.

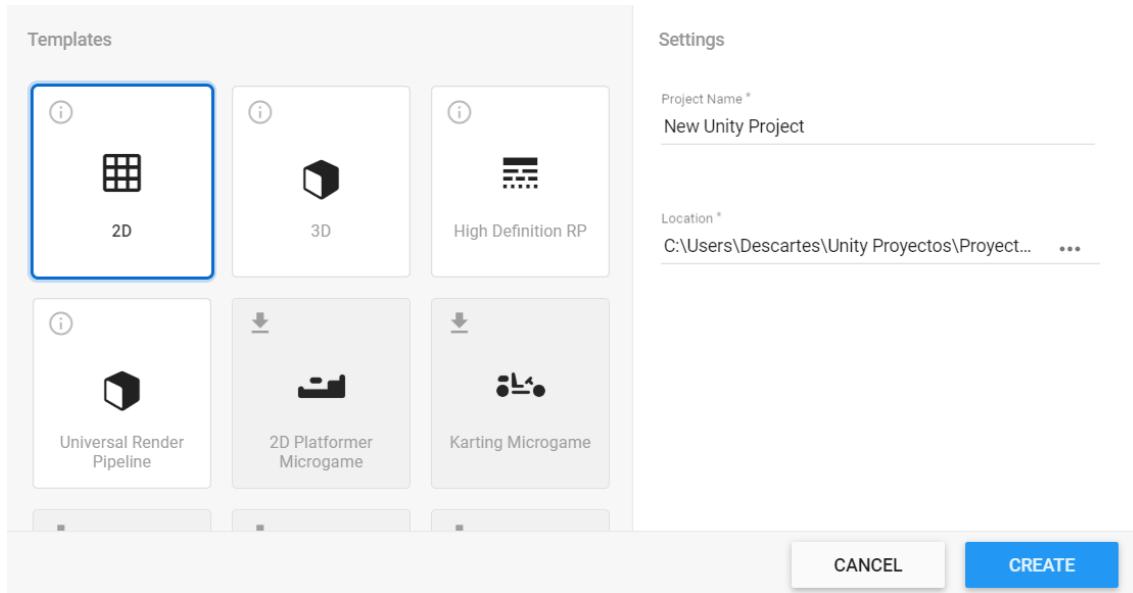
Antes de descargar, elija la versión de Unity que sea adecuada para usted.

[Elige tu Unity + descargar](#)

[Descarga Unity Hub](#)

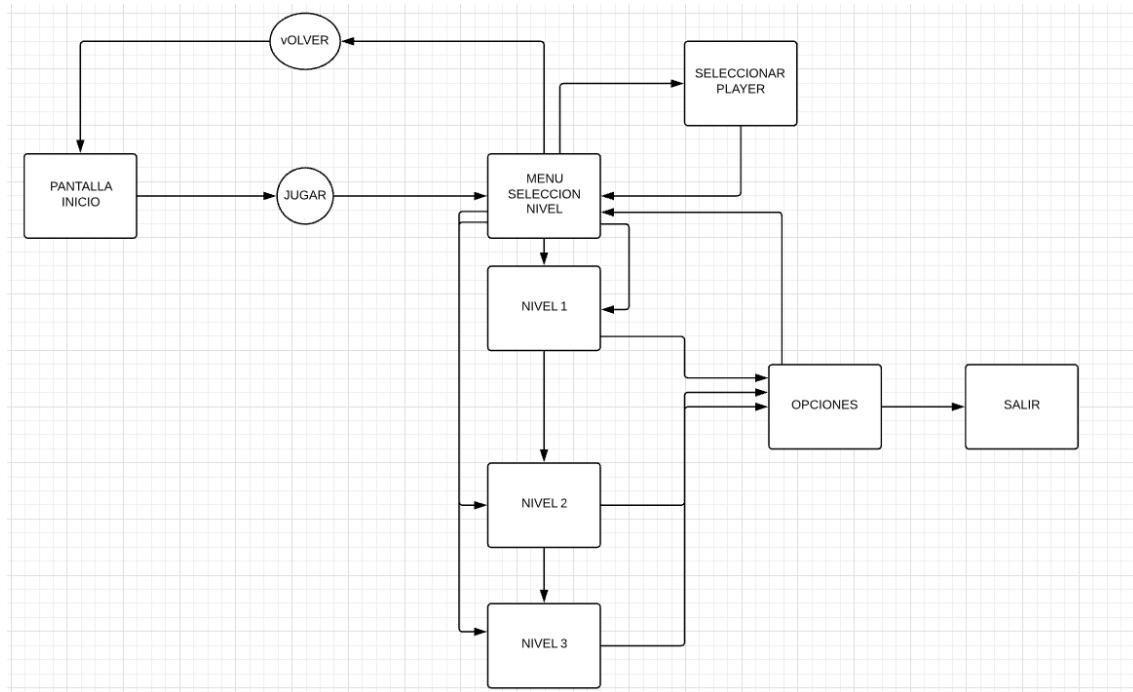
[Descubrir más acerca del nuevo Unity Hub aquí.](#)

El proyecto se inició como Unity 2D.



Como se comentó en distintos puntos de esta memoria y en el propio GDD. El objetivo era hacer un video juego de plataformas en Scroll horizontal en 2D similar a los juegos de la old school, como Super Mario Bros o Donkey Kong.

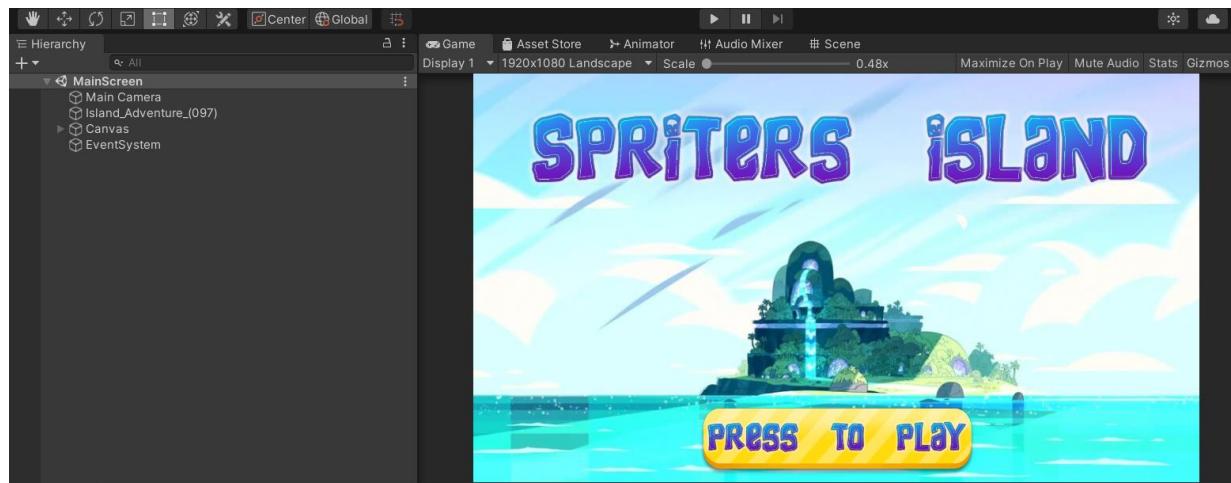
Antes de comenzar a construir el proyecto se diseñó un diagrama que había que seguir exhaustivamente para conseguir el éxito en el citado proyecto.



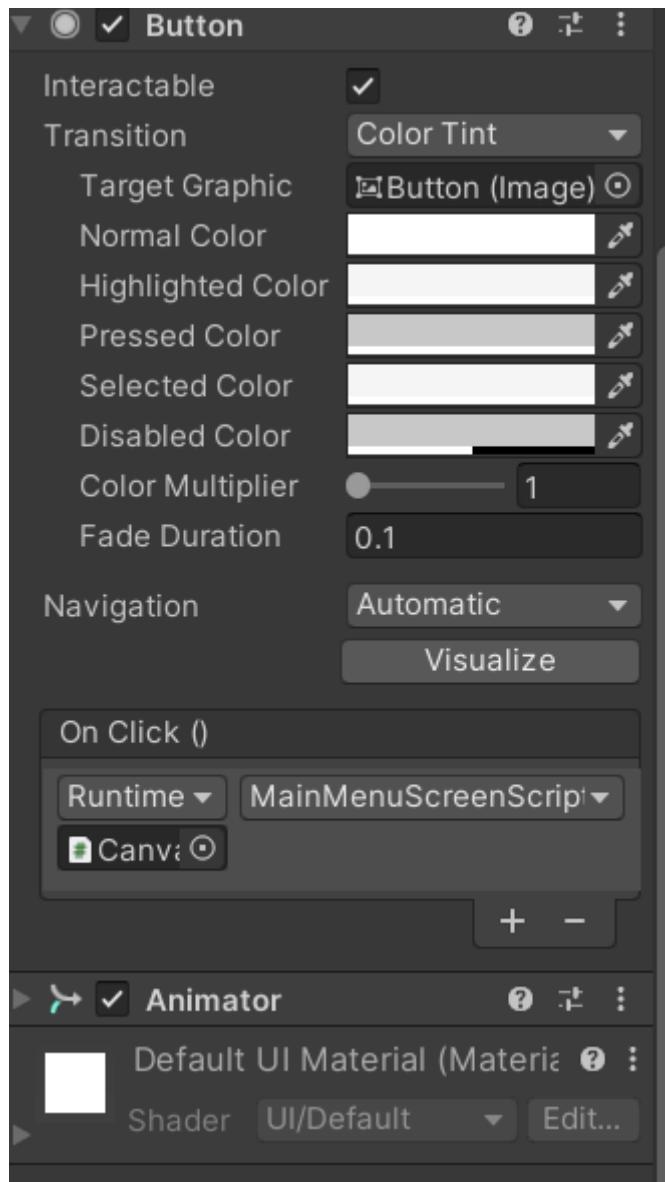
A continuación se mostrará las distintas partes de las que se compone el juego:

Pantalla de Inicio:

Como se puede comprobar el juego comienza con una sencilla pantalla de inicio con un botón de inicio que se encuentra animado ya que aparece y desaparece.



El botón esta programado para que una vez sea pulsado se pase a la siguiente escena, además de su respectiva animación hecha con el animator del motor:



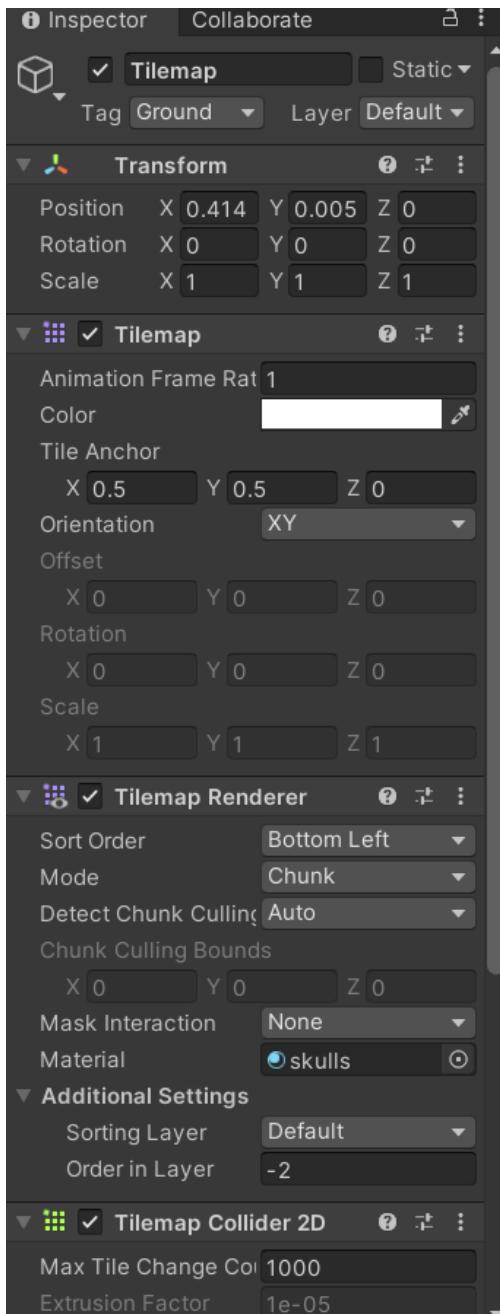
Creando los Stages:

Para la creación de los Stages se han usado 5 Assets distintos que han sido nombrados en otras partes de la memoria y el GDD, estos son: Pixel Adventure 1, 2 y 3 y las 3 variantes existentes de SunnyLand, aparte, se han usado otros assets de forma complementaria obtenidos en otras stores como en Itch.io.

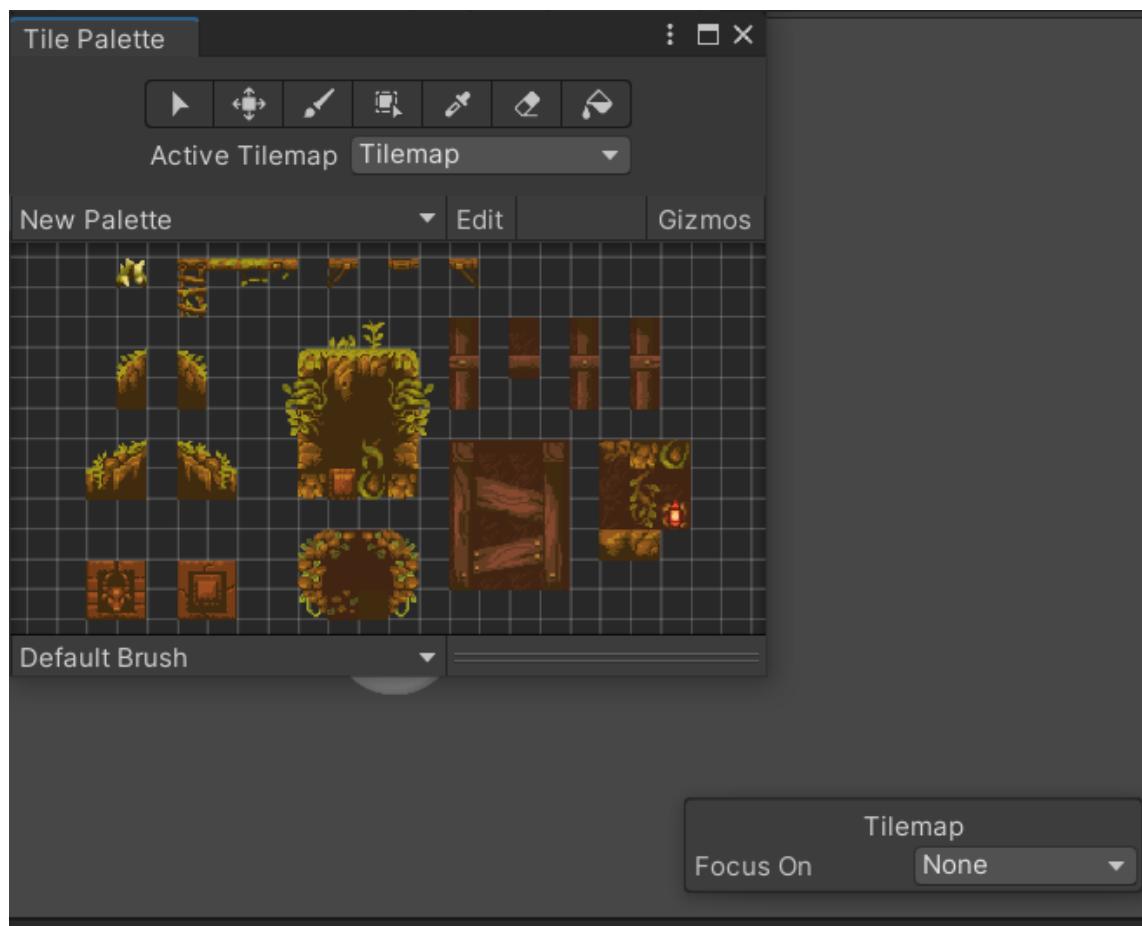
Para empezar a diseñar los niveles primero se creó unos Gameobject llamados Grid, dentro de cada Grid hay otro GameObject hijo llamado Tilemap, esto nos va a permitir poder diseñar el mapa con mayor facilidad.

El motivo de que haya dos es porque uno tiene incluido un collider y otro no ya que habrá cosas que interesen que tengan collider y otras no.

Como se puede ver se le añadió el Tag de Ground a causa del personaje, para que sepa cuando está en el suelo y pueda cambiar su animación.



Este es el Tile Palette que ayuda a pintar el nivel.



El player:

Como se indicó anteriormente hay 7 personajes distintos seleccionables:

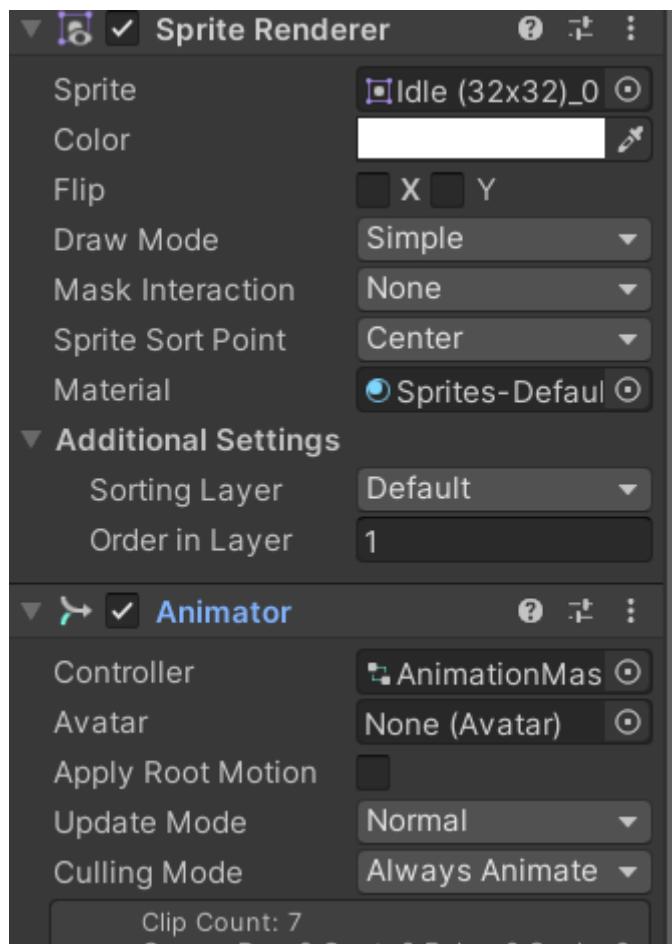


Características técnicas del player:



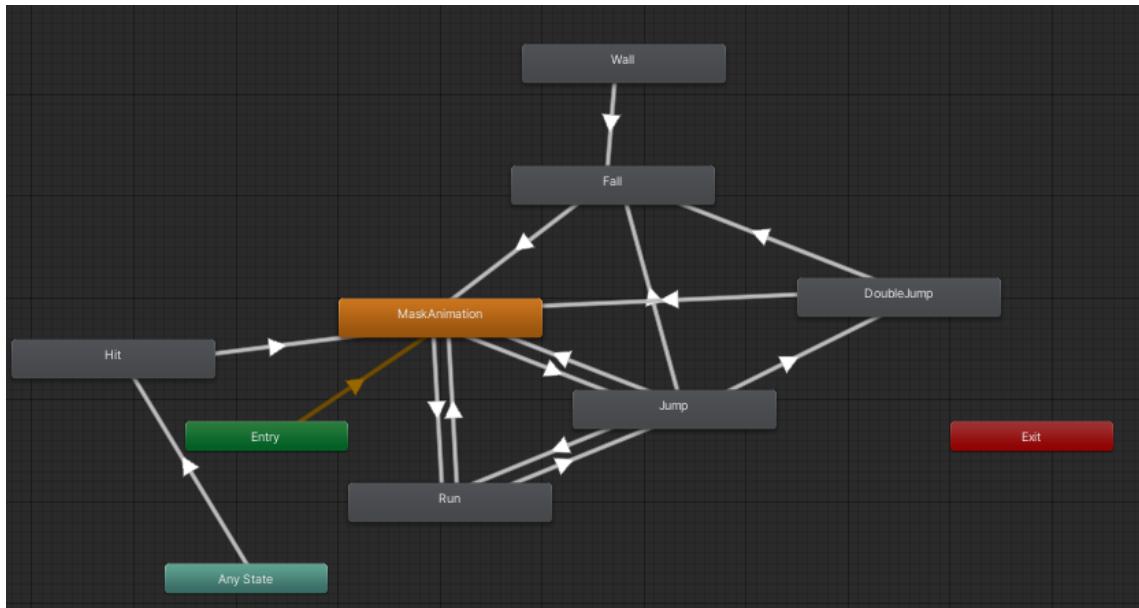
A continuación se procede a detallar todo lo relativo al player:

El Sprite Renderer y el animator es todo lo relativo a la animación del personaje.



El player presenta varios estados de animación:

Este es el árbol de Animación utilizado para que todo lo que procede a exponerse pueda darse lugar:



Idle: Que es una posición “stand” o parada si no se mueve.



Jump: Posición de salto y que vuelve a Idle una vez el player colisiona con el Tag Ground.



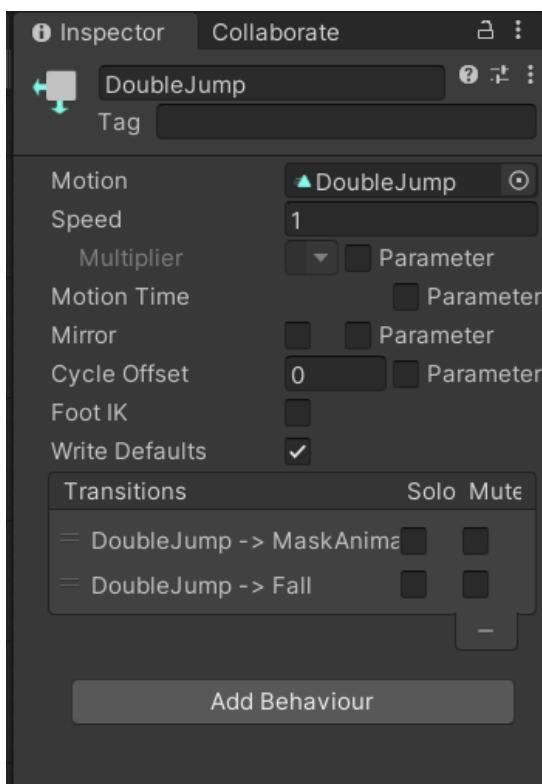
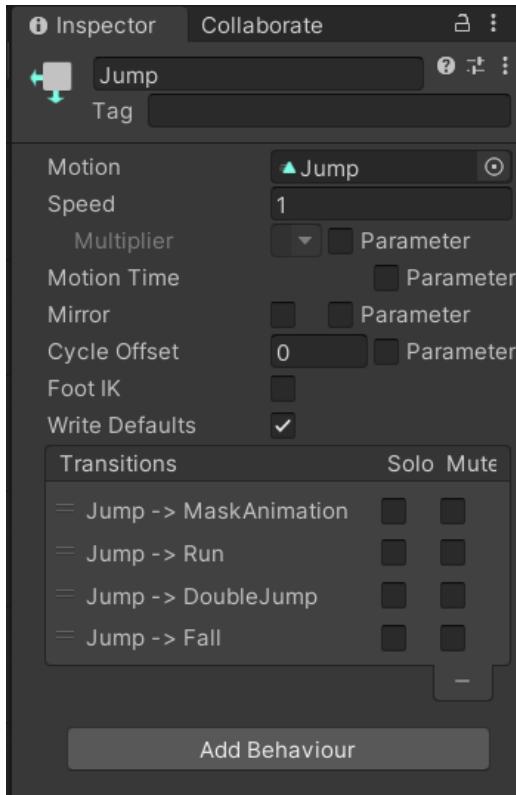
Double Jump: En el aire da un doble salto.



En cuanto al salto se le ha añadido un salto gradual dependiendo del rato que se pulse saltará mas o menos.

Para el doublejump se arrastra los sprites del doublejump y se crea la animación llamada DoubleJump.

Para el tema de la animación del doble salto hay que irse al árbol de animación. Se crea unos parámetros en la animaciones y una condición para pasar al doblejump y se configura como true.



Para la caída del personaje sería mismo. Esta vez es solo un Sprite , se crea la animación llamada Fall.

Luego una vez creada la animación se le da a add property, saldrá la opción de Sprite Renderer y se marca la opción de sprite y así se crea la animación del Fall. Hecho esto se borran los árboles de animación de Fall y Doublejump.

Hurt: Cuando el player recibe daño.

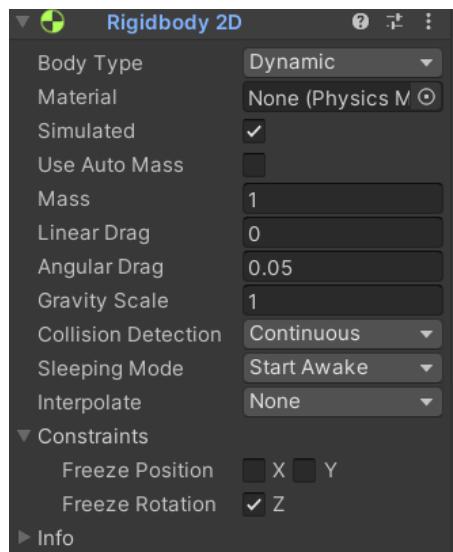


Wall Sliding: Deslizamiento por la pared.



RigidBody 2D:

Esto es toda la parte relacionada con la física del player. En el personaje se tiene que Freezear la rotación en el RigidBody para que el personaje en caso de caerse no vaya rodando.



Player Move:

Este es el Script relativo al movimiento del personaje. Dicho Script se comenta en su apartado correspondiente. Como se puede comprobar aparece la velocidad, el salto, partículas y función de Dash.

También se le ha añadido un salto gradual dependiendo del rato que se pulse saltará mas o menos.

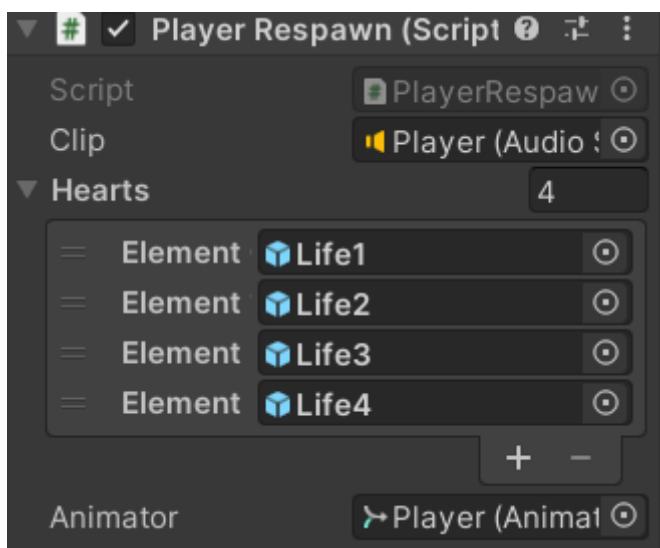


A parte esta la función de movimiento de Joystick habilitada especialmente para Android sobre todo.



Respawn:

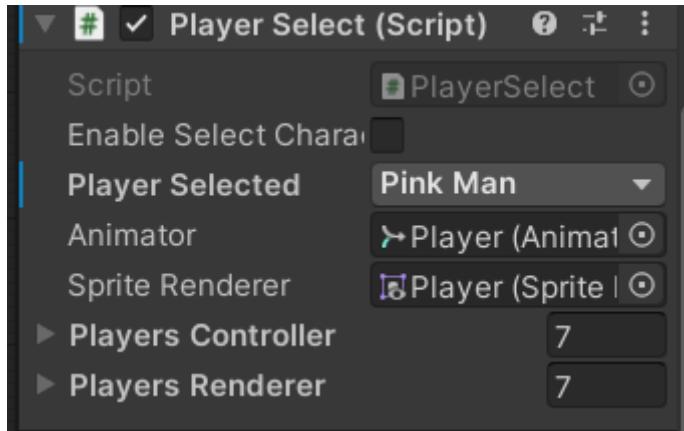
El Script relativo a las vidas del personaje y el reinicio del nivel en caso de perderlas todas.



Player Select:

Para la selección de personaje.

Este es el script para seleccionar personaje en la escena.



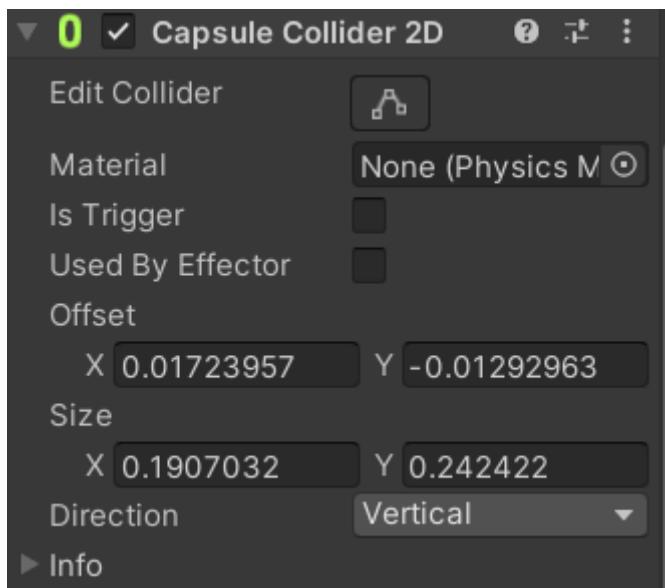
Para añadir los personajes en el juego el procedimiento es el mismo que con el primero, se arrastran los sprites, se crean las animaciones, y se trabajan con las transiciones y con el script del player. Una vez metidas todas las animaciones se borran los arboles de animaciones y se deja solo el idle.

En cuanto al script llamado player select. La forma mas sencilla y simple es modificar el arbol de animaciones es decir el idle y sustituir las animaciones de uno por otro.

Se hará un Script que nos permita dentro de player en Unity elegir un personaje u otro.

El Collider del Player:

Es un colisionador por lo que permitirá detectar cuando personaje choque con enemigo, objeto, moneda, etc.... En virtud de lo que sea el personaje actuará de una manera u otra.



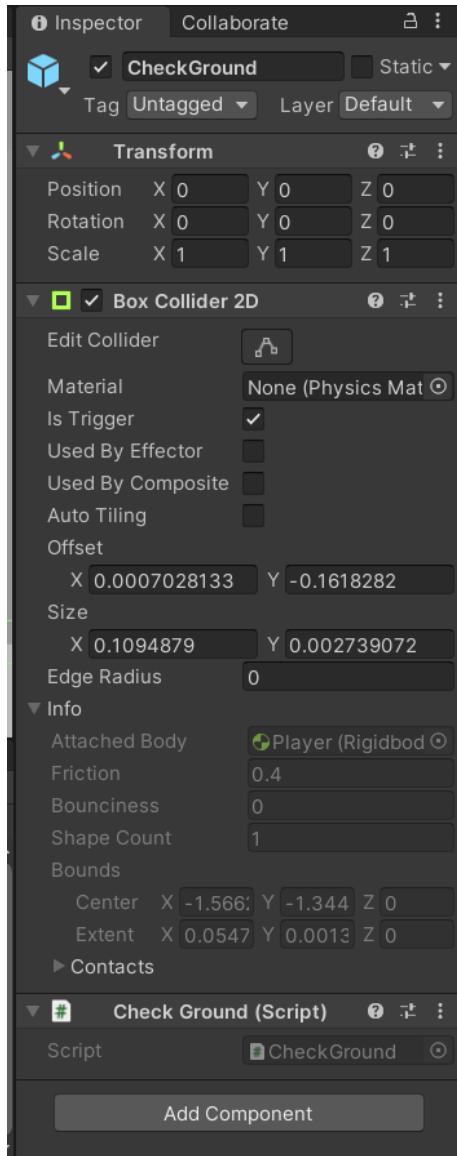


Collider para salto (Checkground):

Para que el personaje salte se crea un Collider, pero es importante saber si el personaje esta tocando el suelo.

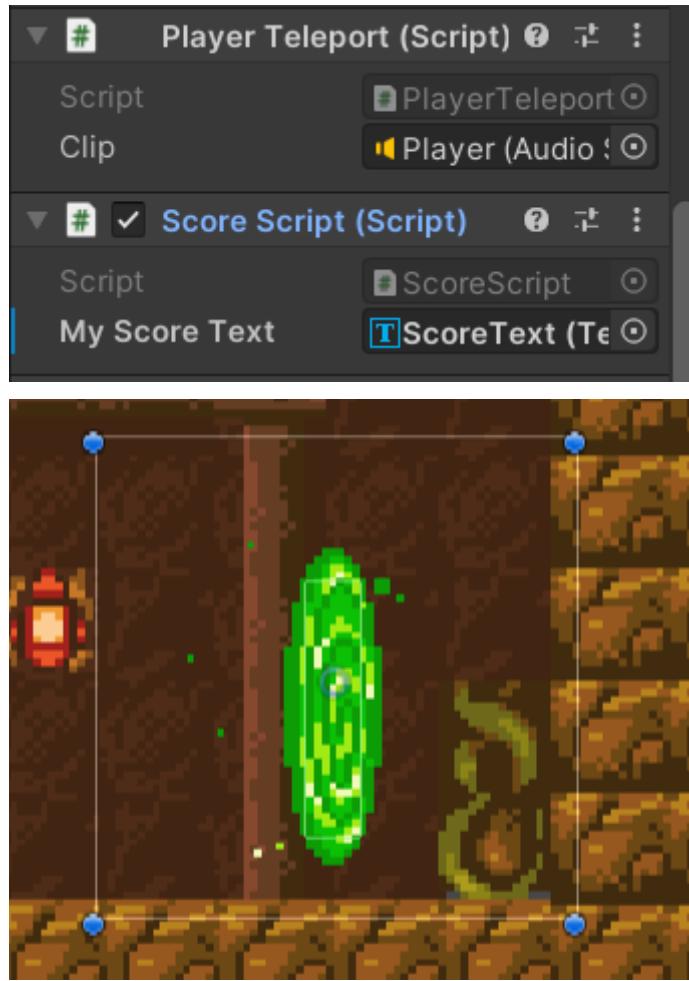
Para ello hay que dirigirse al personaje en este caso MaskedGuy, se le crea un GameObject y se le añade un componente es decir un BoxCollider, se le hace un BoxCollider pequeño practicamente debajo de sus pies. El gameobject se va a llamar Checkground

En el boxCollider se tiene que marcar el Trigger. Así se podrá comprobar que esta dentro del suelo



Scripts accesorios de Score y Teletransportador:

Estos Scripts vinculados al objeto player permitirán poder teletransportarse al player de una dirección y a otra y con el Score Script el player podrá aumentar su puntuación una vez colisione con los objetos como monedas o elimine enemigos.



Para hacer un score se crea un texto y luego el script en unity. En el script se indica que el player colisiona con un tag concreto y luego los programan los if correspondientes y listo.

Enemigos:

Ya se ha hablado de los enemigos en otras partes del proyecto y se ha explicado las características de cada uno. Si es necesario mas información exhaustiva sobre ellos existe la posibilidad de dirigirse al GDD o al inicio del proyecto. En resumen el proyecto se

componía de 10 enemigos (voladores, terrestres, aquellos que lanzan artefactos y perseguidores). En esta parte del proyecto se expondrán como están compuestos cada uno de esos tipos.

Terrestres y voladores:



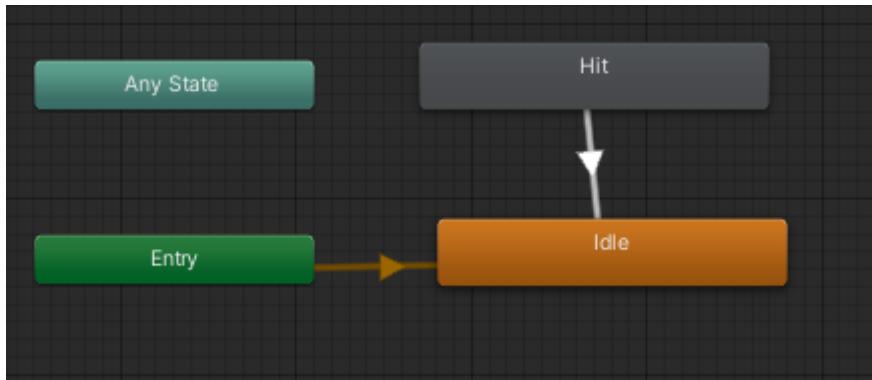
Formado por Mushroom, Blue Bird, Bat, Gator y Eagle:

Estos enemigos se caracterizan porque se desplazan de una serie de puntos a otros. Pueden ir a 2 puntos, a 3 puntos, 4 puntos, etc...

Están formados por un GameObject padre y dentro tiene un GameObject hijo que contiene la información del enemigo, seguido de los puntos de movimiento.



El árbol de animación de estos enemigos se compone de:

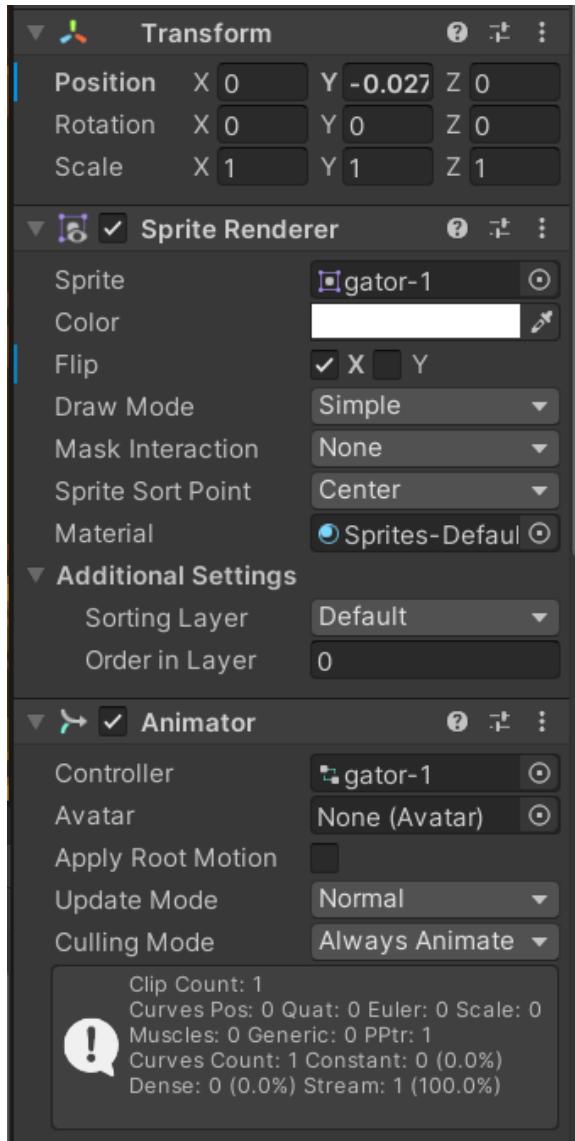


Es decir al ser golpeados vuelven a su estado de Idle.

Se harán una transiciones en estos enemigos de idle a run y viceversa y luego se añade otra transición de cualquier estado a hit porque en cualquier momento se podrá atacar al enemigo.

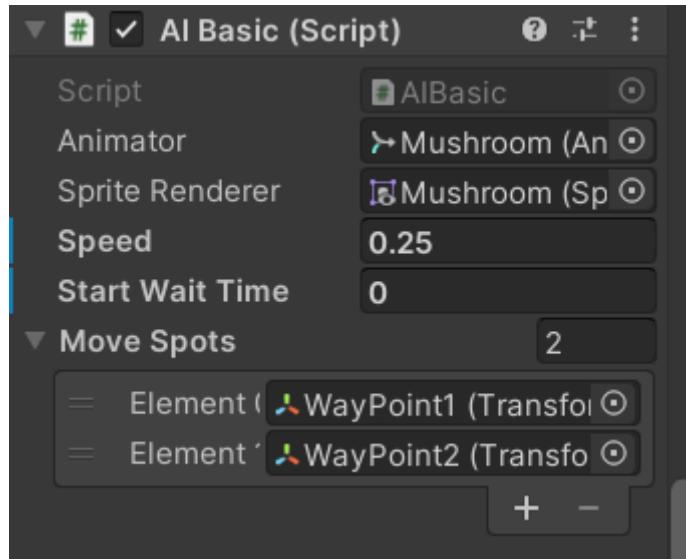
Se crean unas condiciones para que del run pase a idle es de tipo booleano, cuando se este moviendo el enemigo se desactiva el idle y cuando no se mueva se activa

Al igual que el player disponen de lo siguiente:



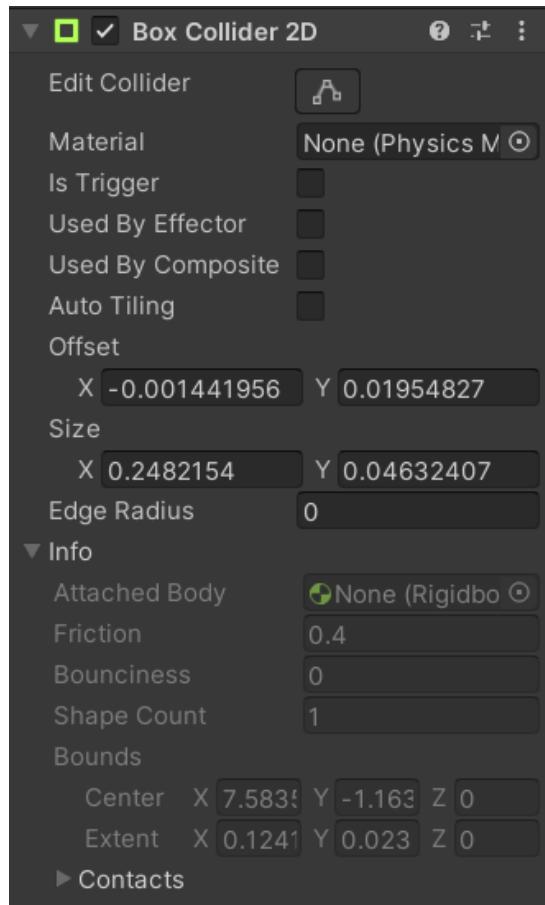
Inteligencia artificial:

Con este script el enemigo podrá moverse de un punto a otro. Con el Move Spots se le indica al Script hasta donde tiene que desplazarse el enemigo y luego una vez allí el siguiente punto.



Collider:

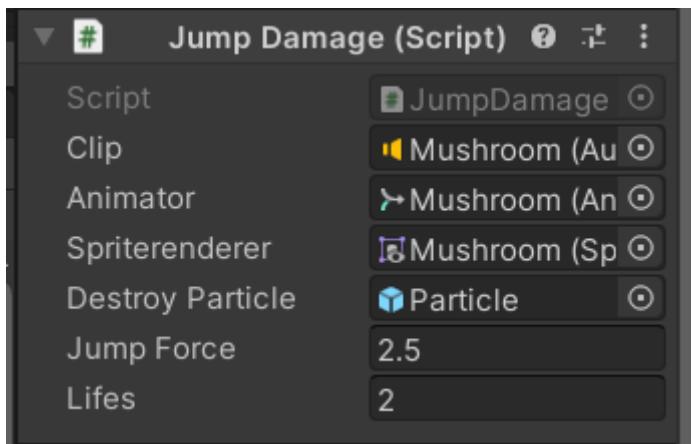
Al igual que el player estos tienen configuradas colisiones por lo que si el player choca con ellos se le restarán vidas al personaje.



Jump Damage:

Este script lo comparten todos los enemigos. Es un collider que en caso de que el player colisione con el se le quitará vida al enemigo o en caso de que no le queden mas vidas desaparecerá.

Aquí se le puede ver el número de vidas que tiene y en caso de saltar encima de ellos el salto que podrá dar el player.



Enemigos que lanzan artefactos:



Compuestos por Plant y Bee.

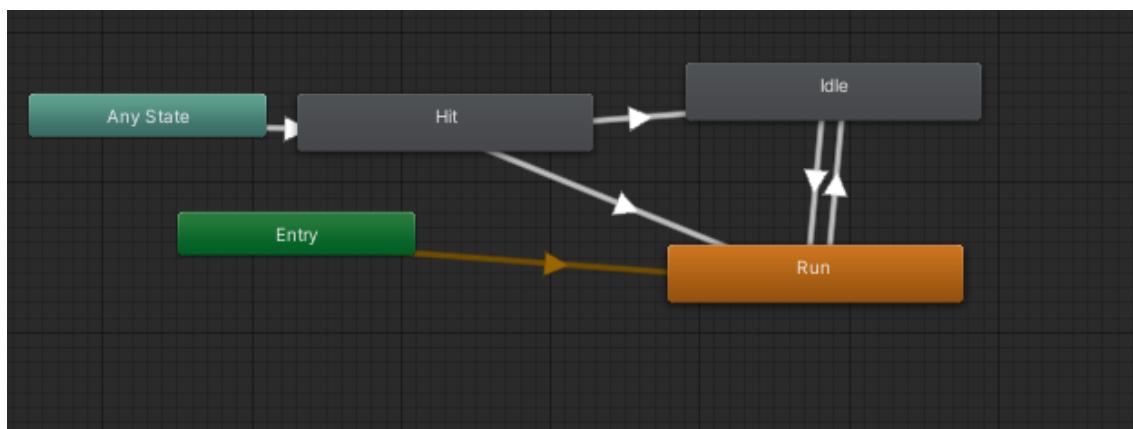
Estos enemigos lanzan aproximadamente cada 3 segundos un GameObject llamado Bullet que si colisiona con el Player le restará vida. Las Bullet se destruyen pasados unos segundos.

Se crean sus animaciones, con el idle y con el resto de animaciones. Se les añade un BoxCollider2d. Se les añade el JumpDamage y se le añade lo que pide. Se le añade también un Enemy Damage para que cuando el Player colisione se debilite. Se le incluye el Jump Collider para que cuando choque el enemigo el player se haga daño. También se le añade el particle.

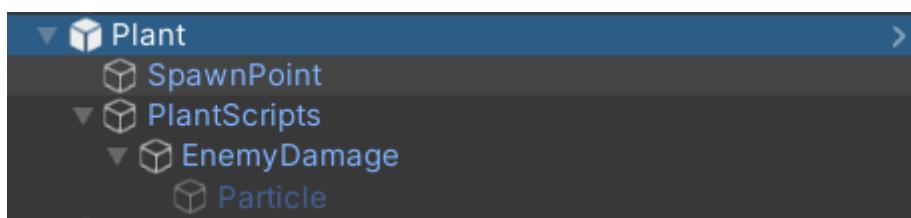
Se crea un GameObject llamado Spawnpoint. Se añade a la planta el Plant Enemy y una vez dentro al Plant Enemy se le asocia lo necesario.

En cuanto a la bala tiene que tener un Circle Collider. Para la bala hay que crear un Script que la bala salga direccionada para ese lado

Mapa de animación de estos enemigos:



Están compuestos por los siguientes GameObjects:

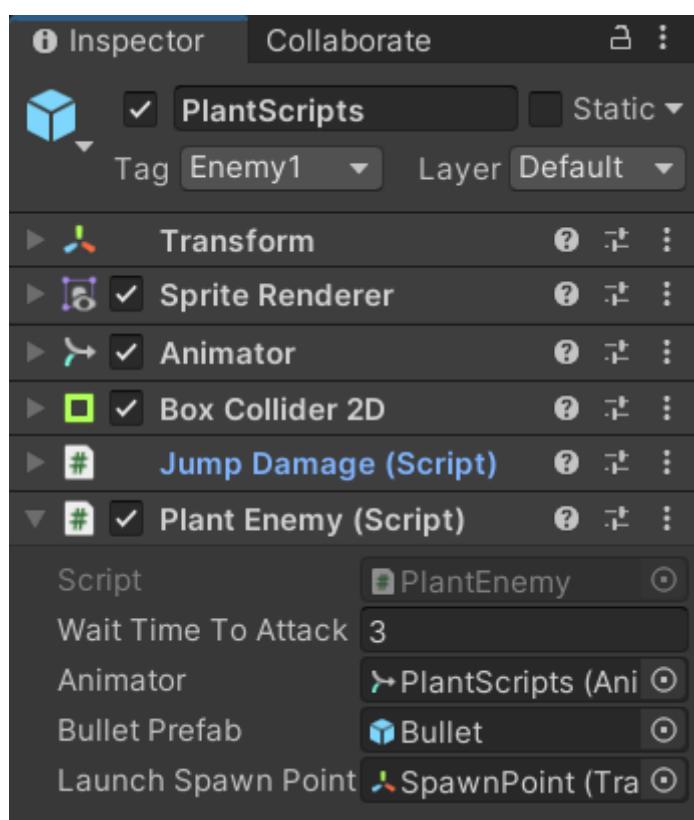


PlantScript:

Este GameObject tiene lo relativo al funcionamiento de este enemigo.

Este enemigo tiene lo que se ha explicado ya anteriormente, un Sprite Renederer, su animator, su BoxCollider 2D para en caso de que el player colisione con él y así le quite vida, el Script de Jump Damage para que el player salte encima de su cabeza y pueda eliminarlo y el Script de Plant Enemy.

En este último script se recoge la configuración de como funciona el ataque de este enemigo. Cada 3 segundos prepara el disparo de una nueva bala y para que este funcione es necesario que se le asocie el prefab de la bala y el Spawn Point que es el sitio desde donde se dispara la bala.



Enemigo Perseguidor:

Este tipo de enemigo te persigue durante toda la pantalla.



Formado por Ghost.

Esta compuesto de los siguientes GameObjects:



El GameObject Ghost se compone de:



Como se puede observar tiene todos los mismos componentes que el resto de enemigos pero esta vez dispone de un Script en el que se le ha indicar su velocidad y el objeto al que ha de perseguir durante toda la pantalla, en este caso el player.

Enemigo con Raycast:



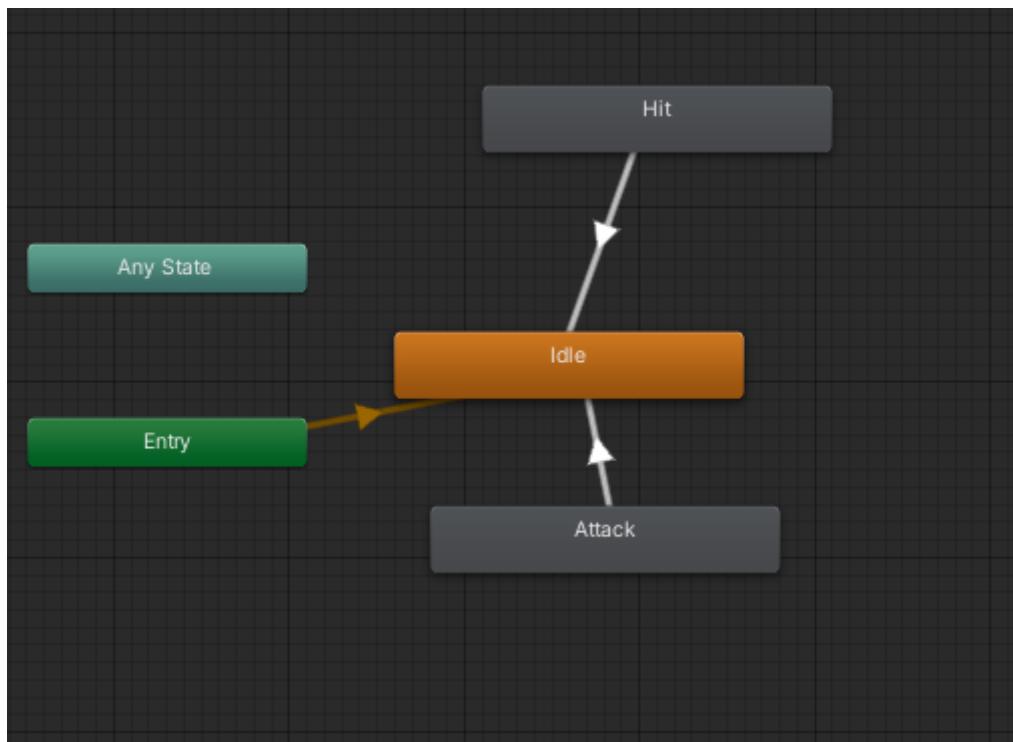
Formado por Zombee.

Este enemigo dispone de un Raycast, es decir, un rayo con el que es capaz de detectar si el player pasa por su zona. Al colisionar el player con el rayo est enemigo lanza su bullet.

Este enemigo se compone de los siguientes GameObjects:

Se desplazará de un lado a otro. Si el player colisiona con el rayo de este enemigo durante el trayecto lanzará el artefacto.

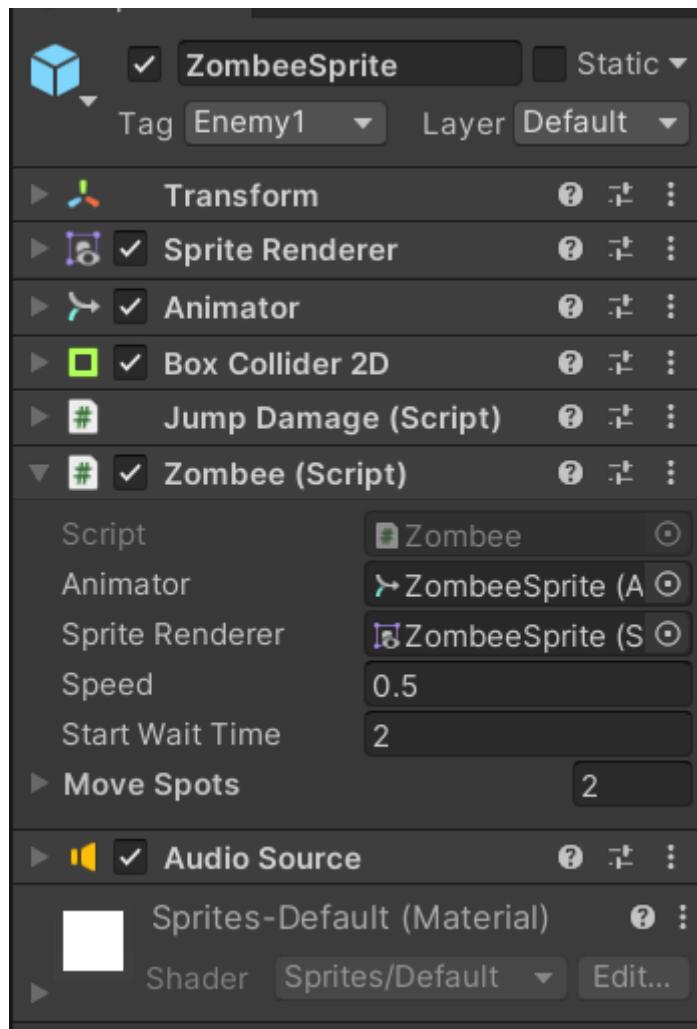
El árbol de animaciones de este enemigo es:



Su GameObject:



Prácticamente se compone de lo mismo que el resto de enemigos:



El Script Zombee indica su velocidad y el tiempo de espera para lanzar su bullet, aparte se puede ver a que direcciones se mueve con el Move Spot.

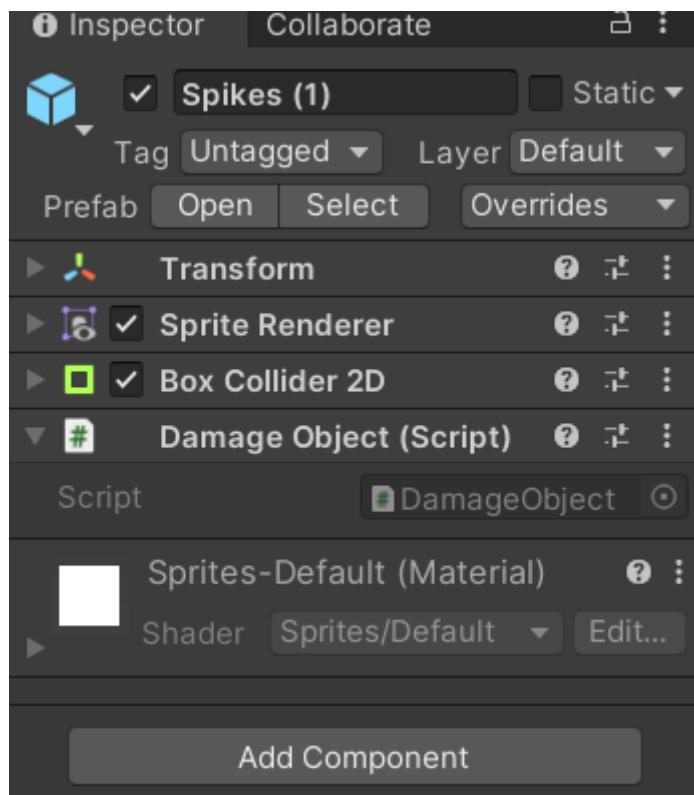
Trampas:

La función de estos GameObject es que cuando el Player colisione con ellos se le resten vidas. Las trampas están compuestas por:

Spikes:



Su GameObject esta compuesto de lo siguiente:



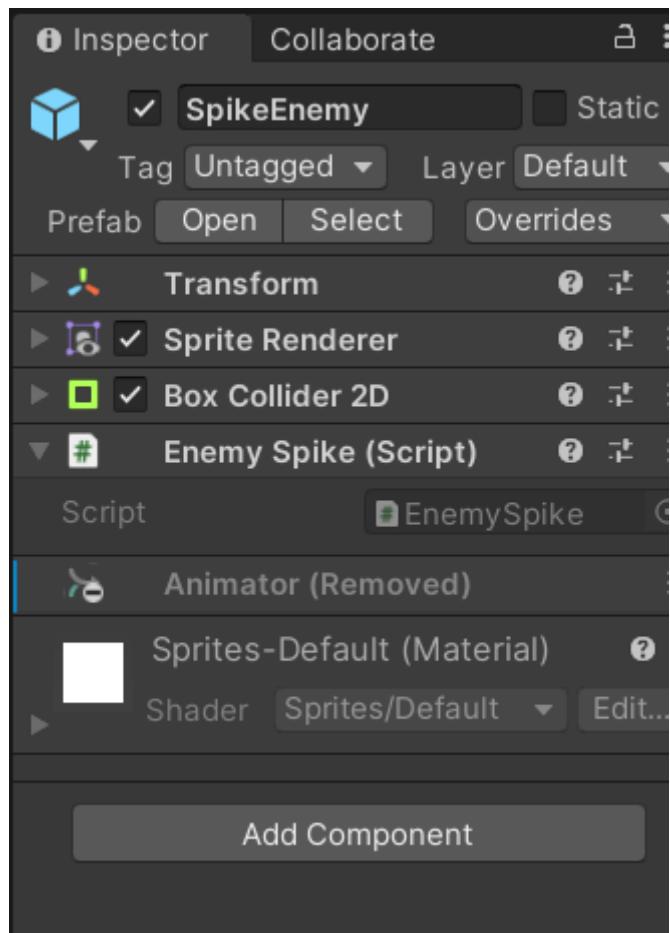
Como se puede ver tiene el Script Damage Object que se ha comentado en varias ocasiones.

Spikes Head:



La característica de este enemigo es que encuentra configurado con Animation por lo que este es capaz de subir y de bajar en función del tiempo que se le haya configurado.

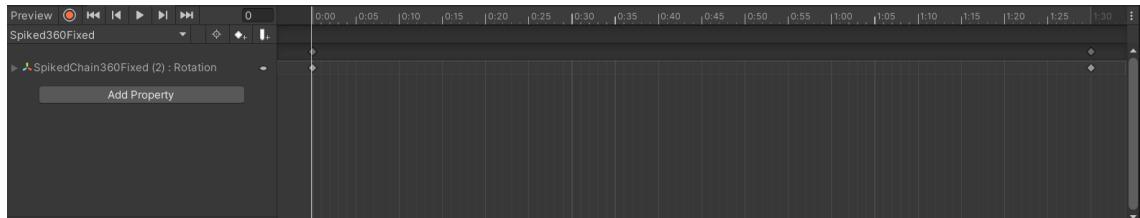
Tiene una composición similar al anterior:



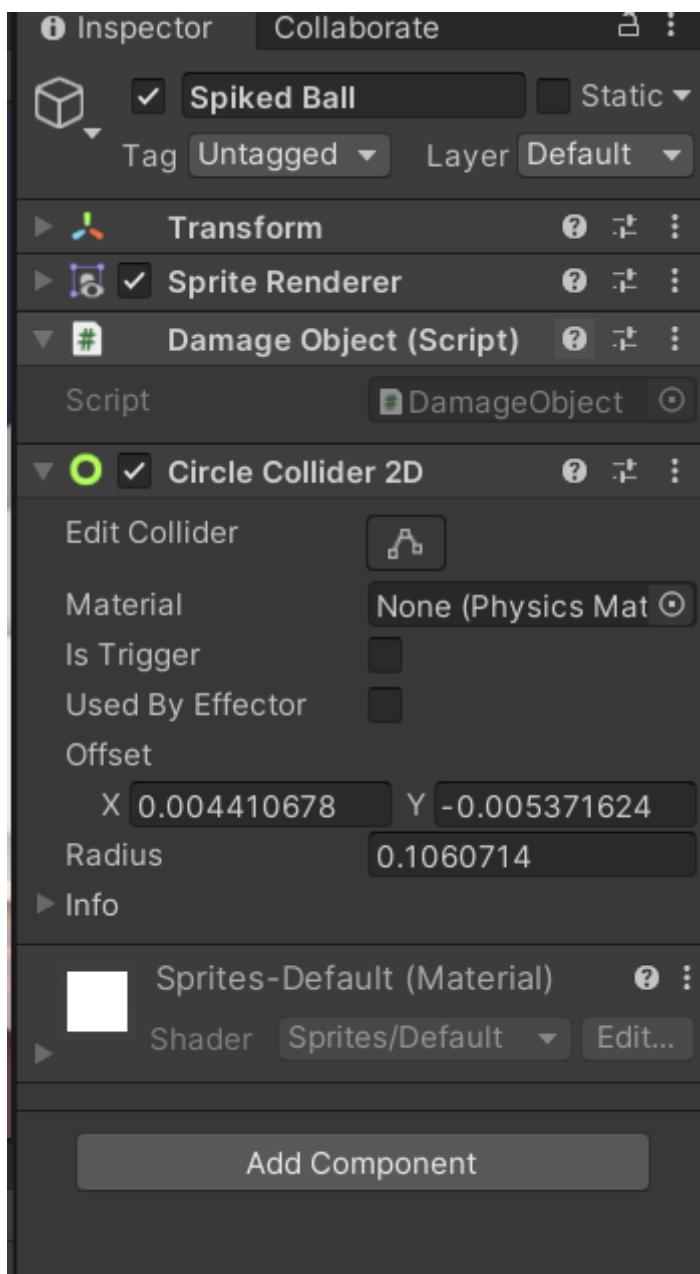
Spiked Ball:

Este GameObject tiene la posibilidad de girar 180 grados o 360 grados gracias a la configuración en su animation.





Una vez configurado el animation se procede a gestionar el interior del GameObject. Esta compuesto del resto de componentes y Script que el anterior salvo que esta vez tiene un Circle Collider.



Como se hizo esta trampa:

Hay que animar un Empty Object cogiendo todos los elementos, el ultimo chain es el punto de pivote.

Se creó un GameObject que contiene todo llamado SpikedFather para que la bola vaya mas fluida en su animación, en curves se seleccionan sus puntos y se le marca la opción de auto.

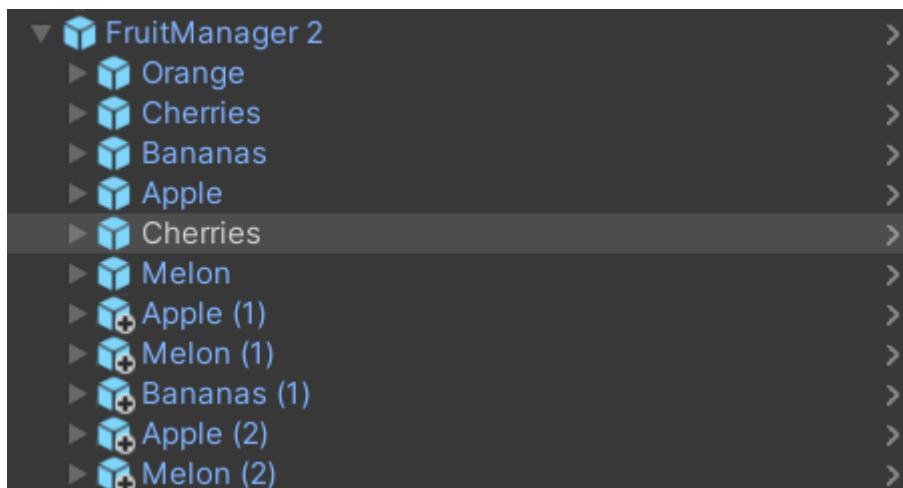
Para la bola 180, en los animator, se configura la rotacion a 90 y -90 y en la bola 360, se configura la rotación a 360, se incluye el damage object y un circle collider.

Frutas:

Uno de los ejes principales del juego.

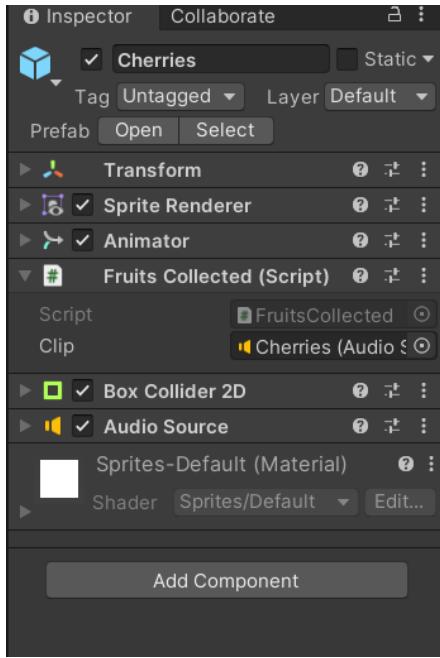


Son GameObjects hijos que se encuentran dentro del Fruit Manager:



Las frutas están compuestas de lo siguiente:

Tienen un Collider para que el player detecte cuando colisiona y un Script llamado Fruit Collected que cuando el player detecta que colisiona estas se destruyen.

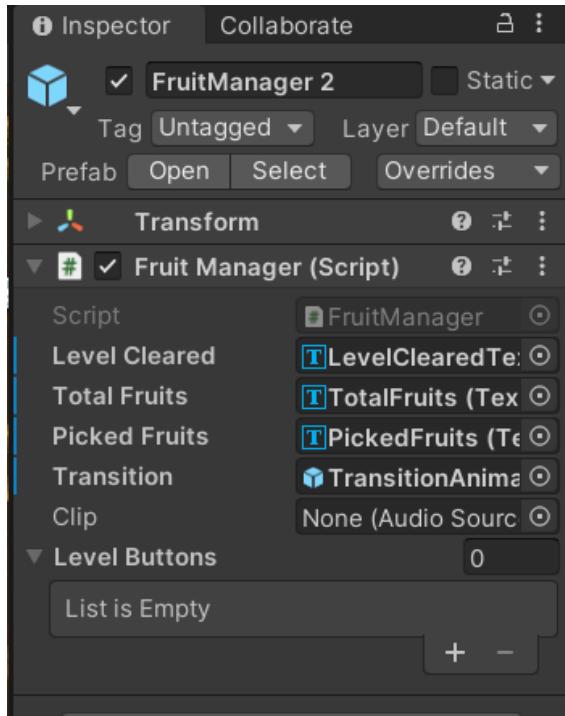


Para que las frutas se recojan y tengan su respectiva animación de haberla recogido se crea la animación de recoger las frutas monedas etc.... ahora hay que añadir a las frutas en cuestión un BoxCollider2d que se va a poner en Trigger y ahora el collected que se ha creado le vamos a añadir un hijo en la fruta.

Para la recogida de la fruta hay que desactivar el sprite renderer cuando se coja.
La intención es que cuando se desactive el sprite renderer se desactive la animación para así ver que lo se ha recolectado.

Fruit Manager:

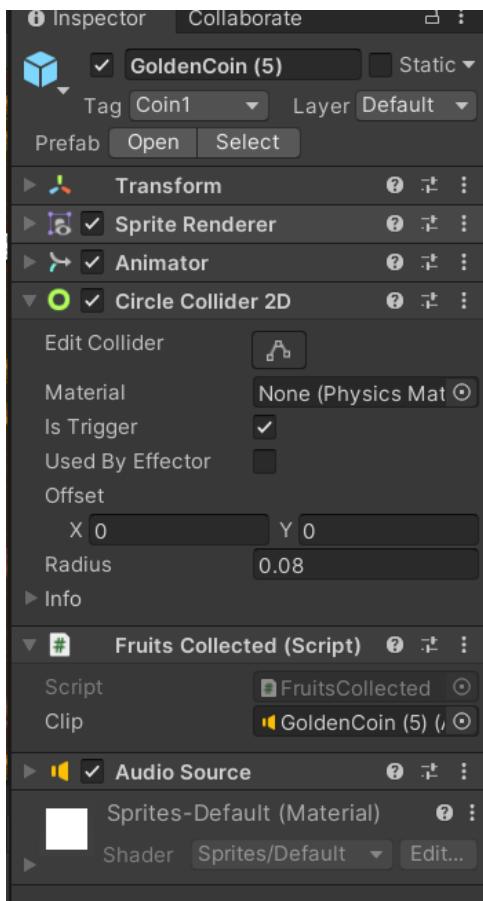
Es el gestor de la recogida de la fruta. Es el GameObject Padre de la recogida de las frutas. Este va contando el número de frutas que hay y cuando el player colisiona con una desaparece y se destruyen también del GameObject padre. Una vez que no hay ninguna el script esta programado para poder pasar de fase o cambiar escena.



Monedas:



Tienen una programación similar a las frutas, es decir, cuando el player colisiona con ellas desaparecen, en este caso se aumenta el Score, pero es debido a que tienen el Tag de Coin por lo que el script del Score detecta que cuando el player colisiona con el Tag Coin y se aumenta dicha puntuación.

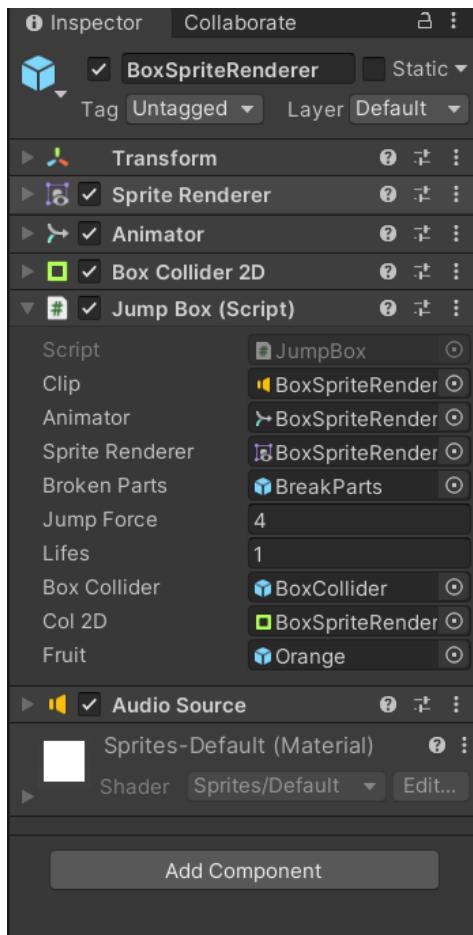


Cajas:



Para la caja se crean unos GameObject. Un gameobject padre llamado Box donde se va a meter el resto de elementos de los hijos. Se va a poner un BoxCollider a la caja. Tanto para saltar etc...

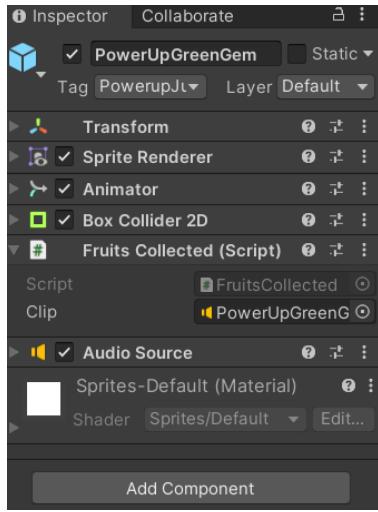
Se va a crear otro GameObject con las piezas rotas de la caja donde una vez la caja tenga 0 vidas o menos saldrán despedidas mediante un script.



Power Up (Gemas):

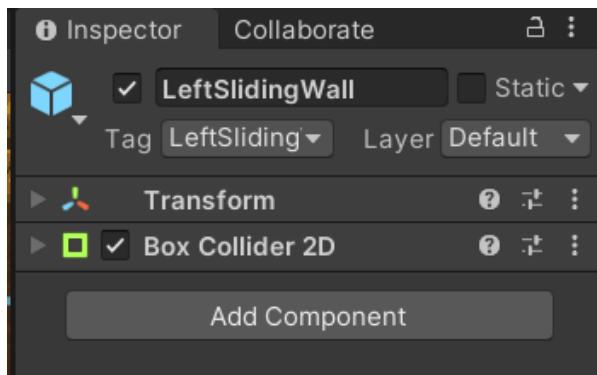


GameObjects en los que el player al colisionar con ellos cambia durante 10 segundos su configuración volviendo una vez pasado el tiempo a su estado inicial.



WallSliding:

Para hacer las paredes deslizantes hay que poner un collider en específico. Se crean las paredes y unos GameObject, uno será la pared izquierda y otra la pared derecha, se le añade un boxcollider a cada uno. Se le crea un tag a cada pared. Creamos la animación del WallSliding. En el animator del player añadimos el wall, le ponemos la transición del wall al fall y le ponemos una condición, cuando falling sea true.



Trampolín:



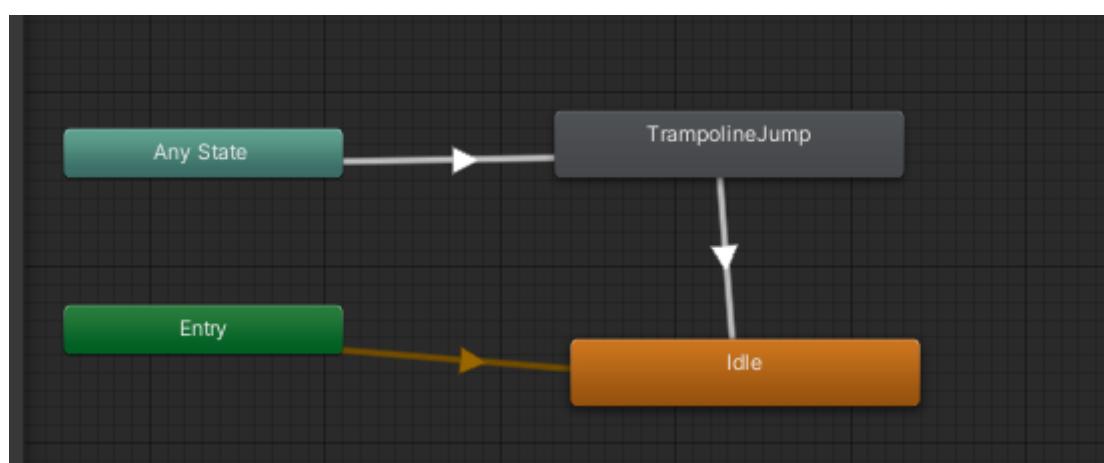
Se crea un trampolin para el personaje. Se coge el Idle del trampolin y se le crea una animación, también se programa un Script para el trampoline.

Para ello se le añade un BoxCollider para que detecte cuando el player colisiona con el.

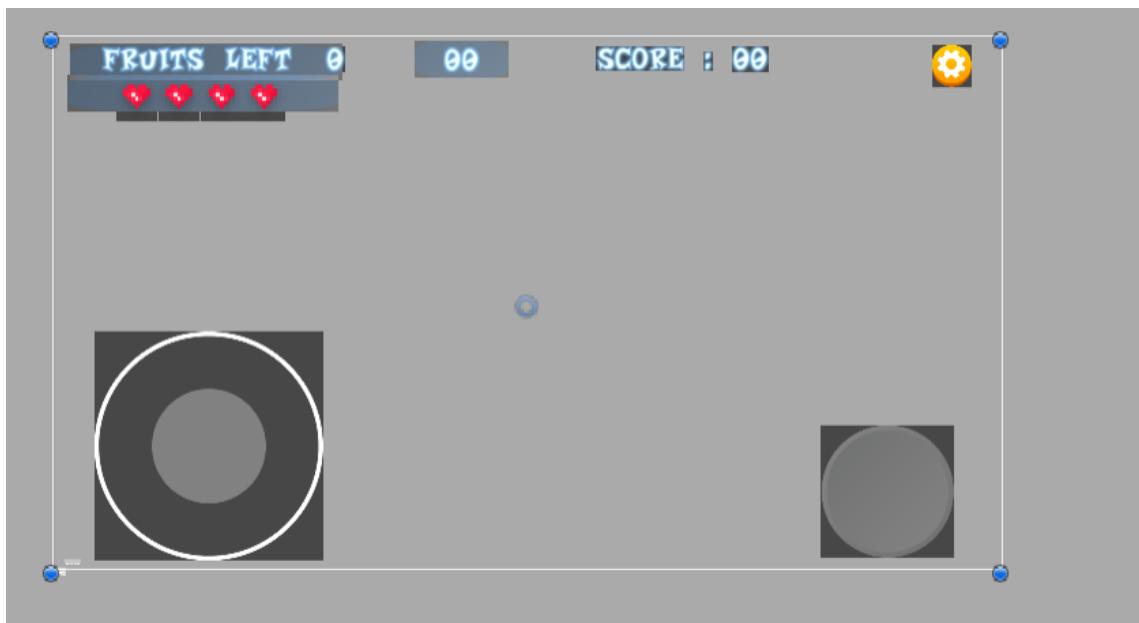
Ahora se configuran las animaciones del trampolín. En el animator se mete el TrampolineJump. Como se va activar en cualquier momento se le hace una transición de any State a Trampoline Jump y de TrampolineJump a Idle.



Árbol de animación:



Canvas:



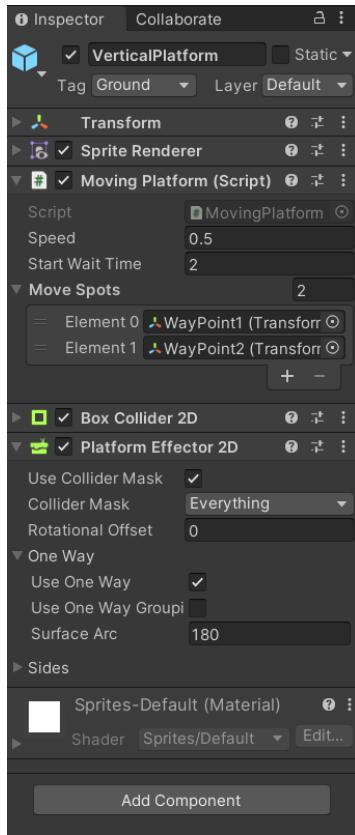
Elemento muy importante del proyecto donde se recogen las vidas, el timer, las opciones y los controles a nivel grafico para Android.

Plataforma en movimiento:



Para hacerla se crea un GameObject que será el padre y dentro se tendrán a los hijos que será la plataforma y los puntos de dirección donde ira la plataforma. Para la plataforma en movimiento se le añade un BoxCollider. Se le añade el platform efecto para que el playe pueda incorporarse desde abajo

El player no se mueve bien dentro de la plataforma por lo que hay que hacer una serie de cosas para que la plataforma detecte que es hijo suyo y se quede adherido.

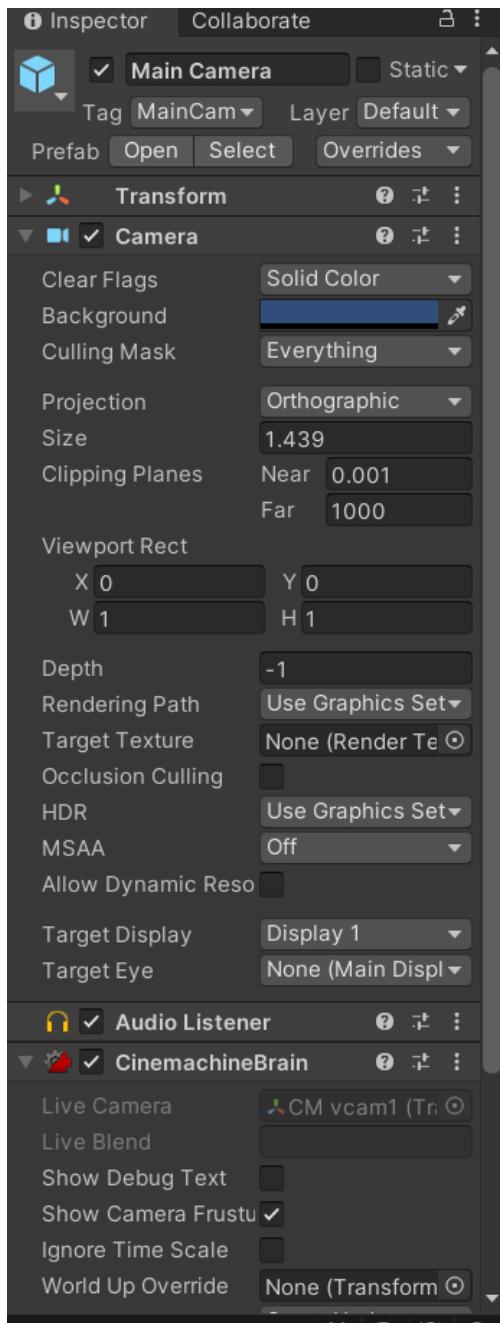


Cámara:

En cuanto a la camara se le añde un Cinemachine. Se le instala Cinemachine y se crea una camara 2d.

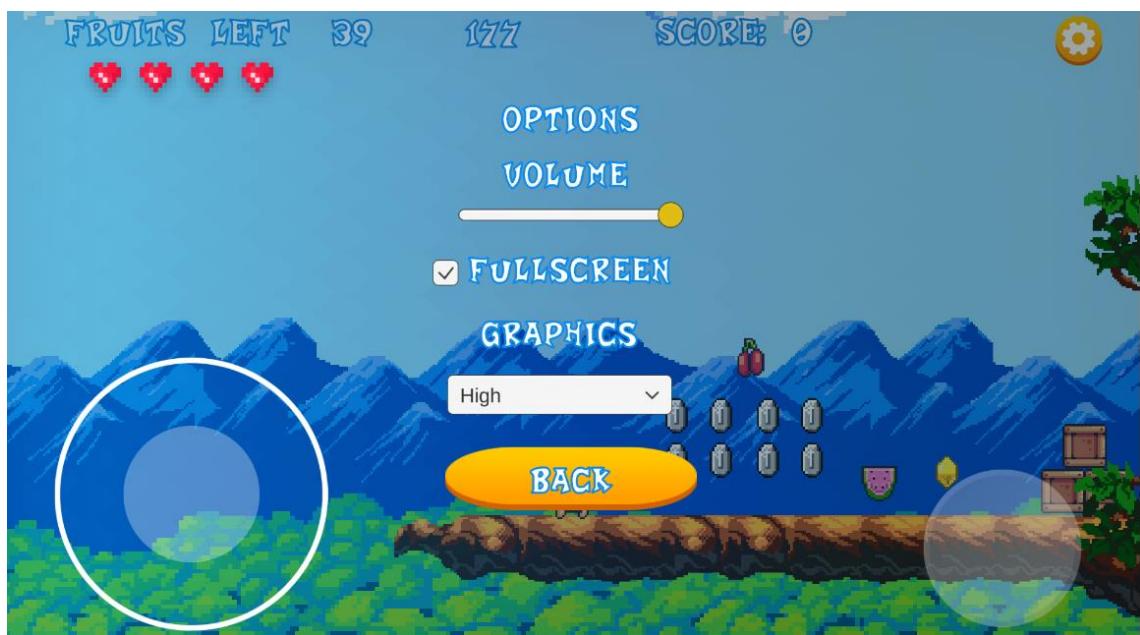
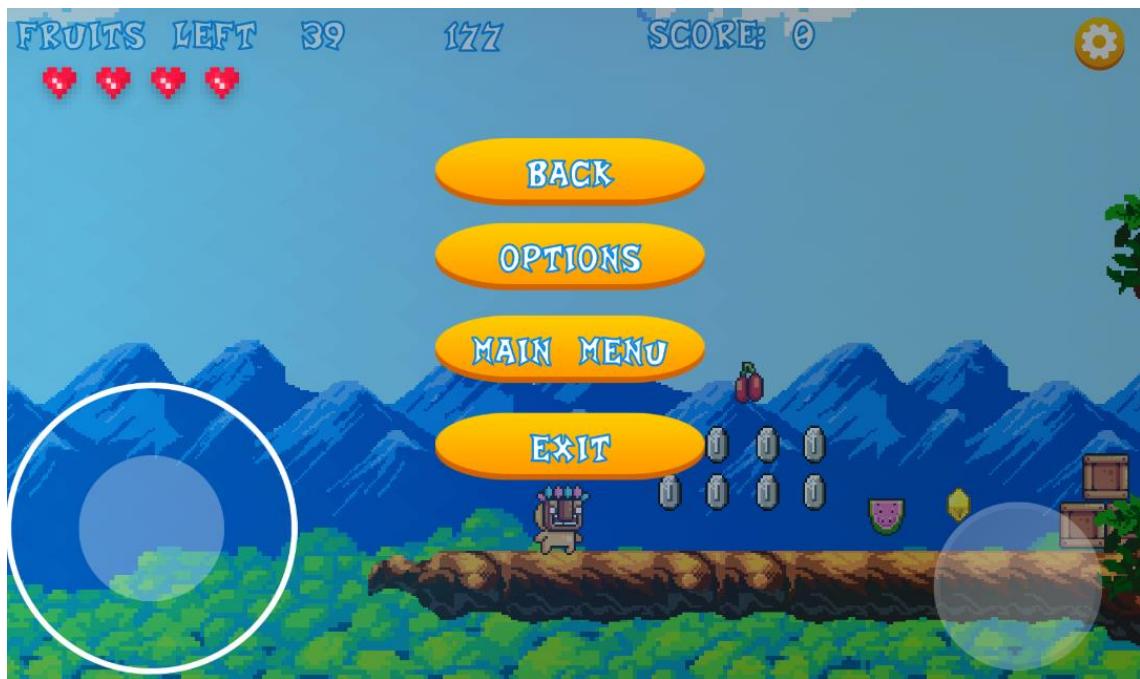
Se le añade la extensión de Confiner. Para que cuando se mueva la cámara no se vea espacio que no se quiera ver, se crea un objeto llamado MapCollider.

Para ello se le añade el Poligon Collider2d con el que se delimitan las zonas que no se quieren que se vean.



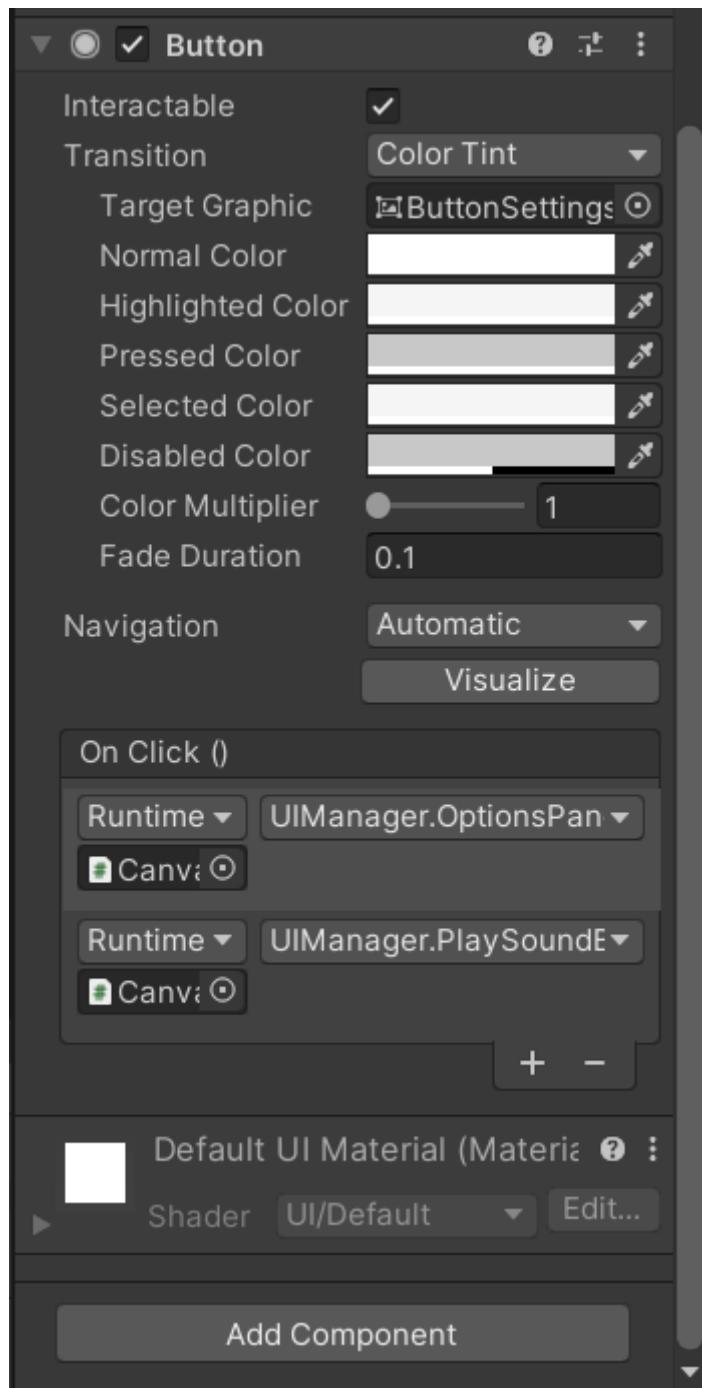
Opciones:

Se ha creado botón que servirá como pausa, opciones, vuelta al main menu. Para ello se hizo una serie de botones con diversas funciones.

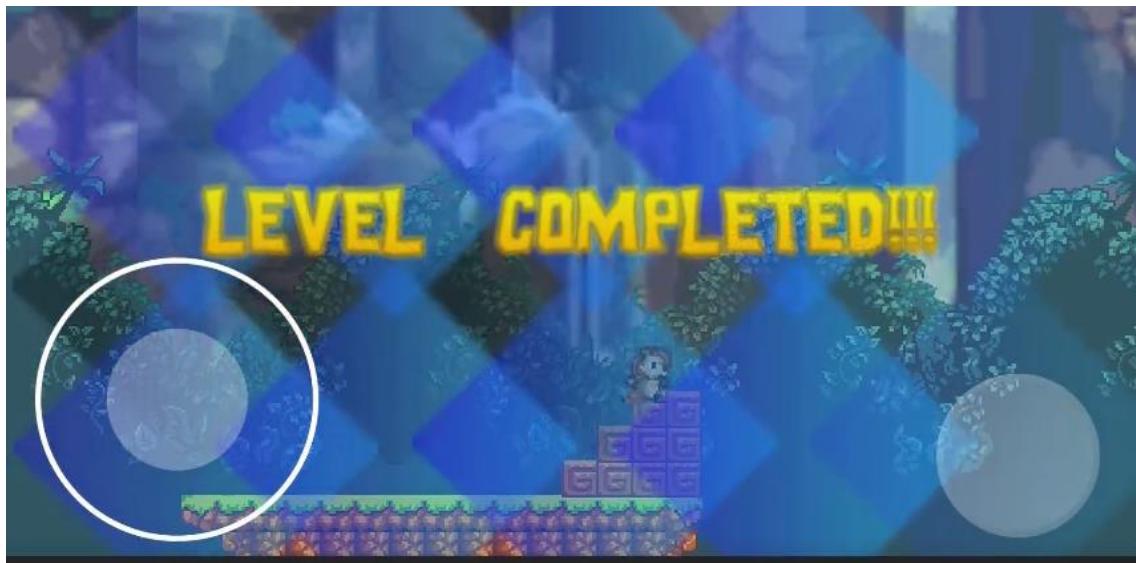


La cuestión importante en el apartado de las opciones está en la gestión de los botones.

Una vez se pulse desaparecerá este menú de opciones debido a un evento on click configurado y aparecerá otro. Todo esto va por Script.



Transición:



Para la transición cuando se pase de nivel se crea un objeto llamado Transition Animation.

Se cogen los sprites de Animation y los colocamos. Seguidamente se crea una animación y le da a grabar

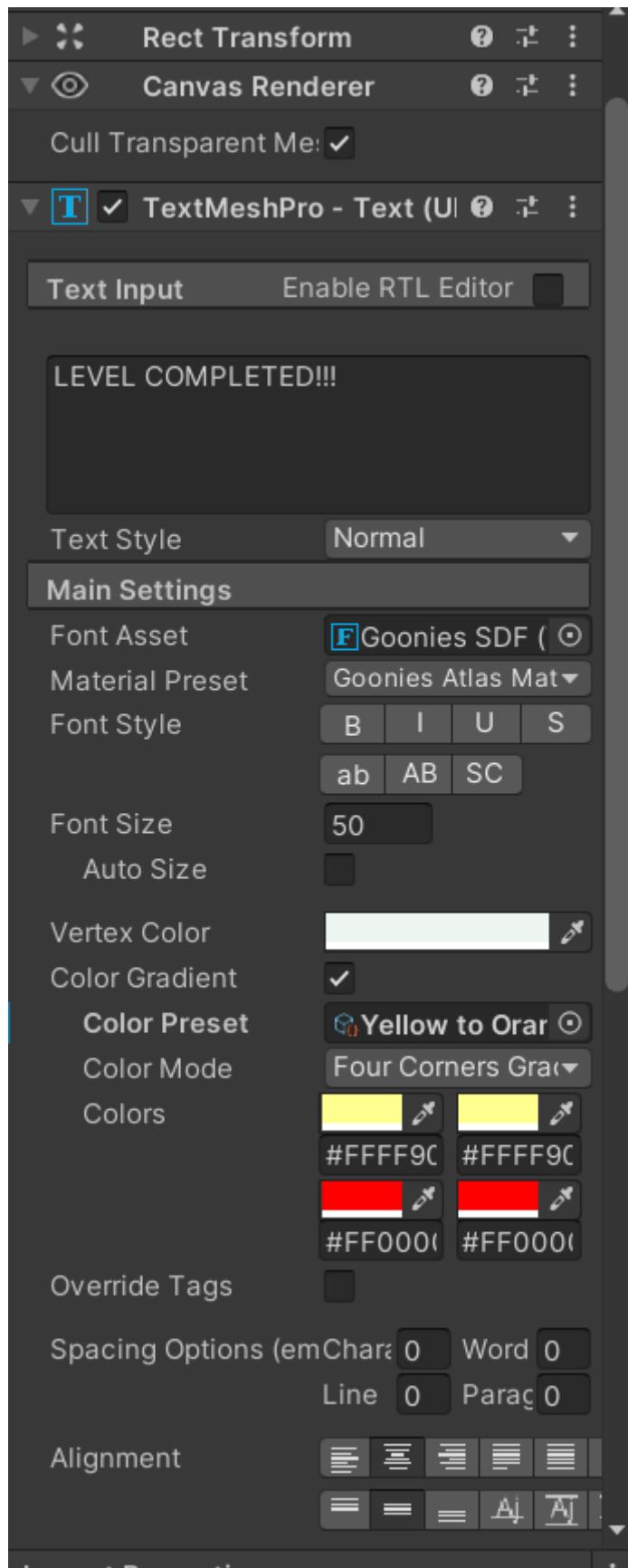
Se pondrán a la escala a 2 y el Alfa 0. Hay que irse al segundo 30 y se la cambia la escala a 4.

En el animator se hará que cuando pasen los segundos vayan apareciendo los sprites poco a poco, usando el botón de rec y seleccionando cada imagen y configurándola en el segundo correspondiente para que vayan acorde.

Para que funcione la transición se añadió en el script del Fruitmanager un public gameObject transition que se activa una vez recogidas las frutas.

Volviendo a Unity se podrá comprobar que el campo transition del FruitManager se encuentra en blanco, por lo que hay que asociarlo con el transition animation.





Partículas de polvo:

Se le han añadido al player unas partículas de polvo por lo que cuando andan son mostradas y da sensación de movimiento.

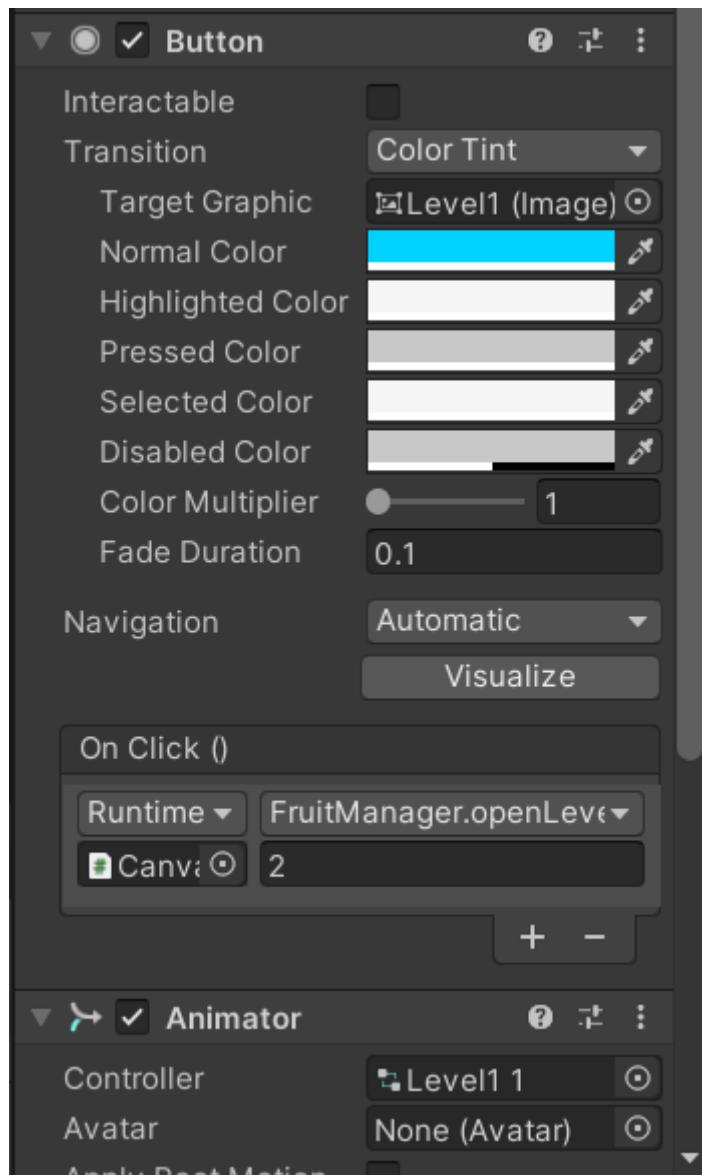
Se incluye el dustparticle y se crea un GameObject como padre de este dust particle. Se hará lo mismo para la izquierda y la derecha. Ahora se le da animación a la partícula de polvo.

Escena de Selección de nivel:

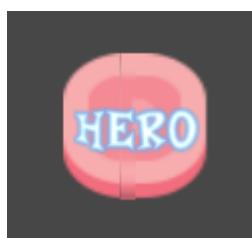


Tal y como se ha expuesto anteriormente son niveles desbloqueables. Los botones de selección son animados y se mueven de arriba abajo. Este efecto se ha conseguido con el Animation.

Estos botones están configurados con un Event on Click por lo que dependiendo donde se pulse harán una acción u otra.



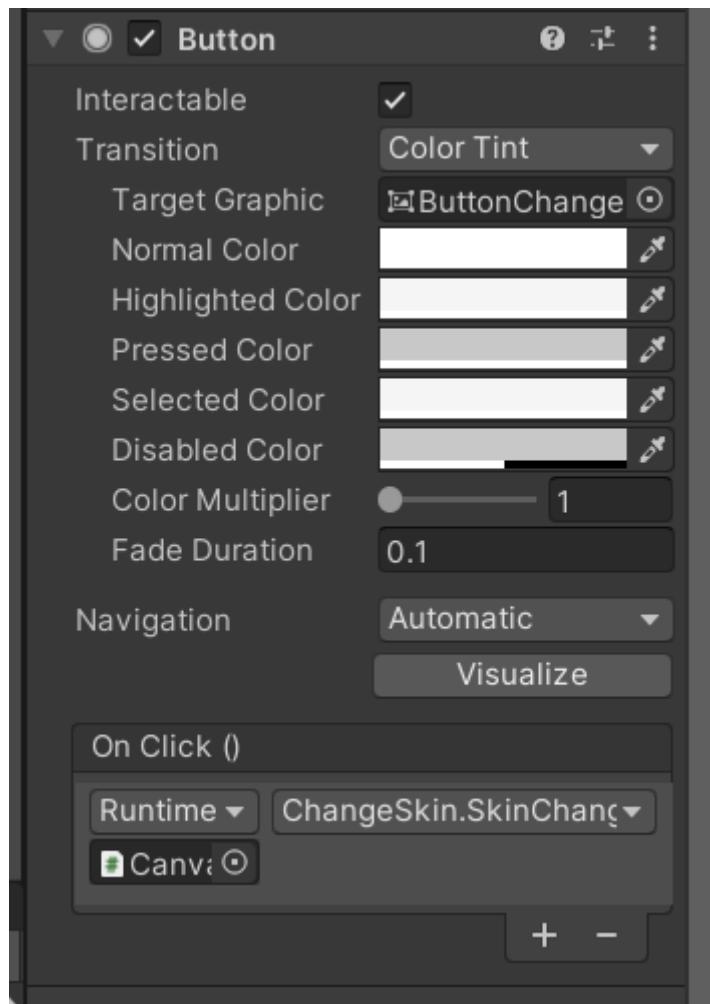
Selección de Héroe:



Este botón mostrará otro menú de opción desplegable y podremos elegir Héroe.

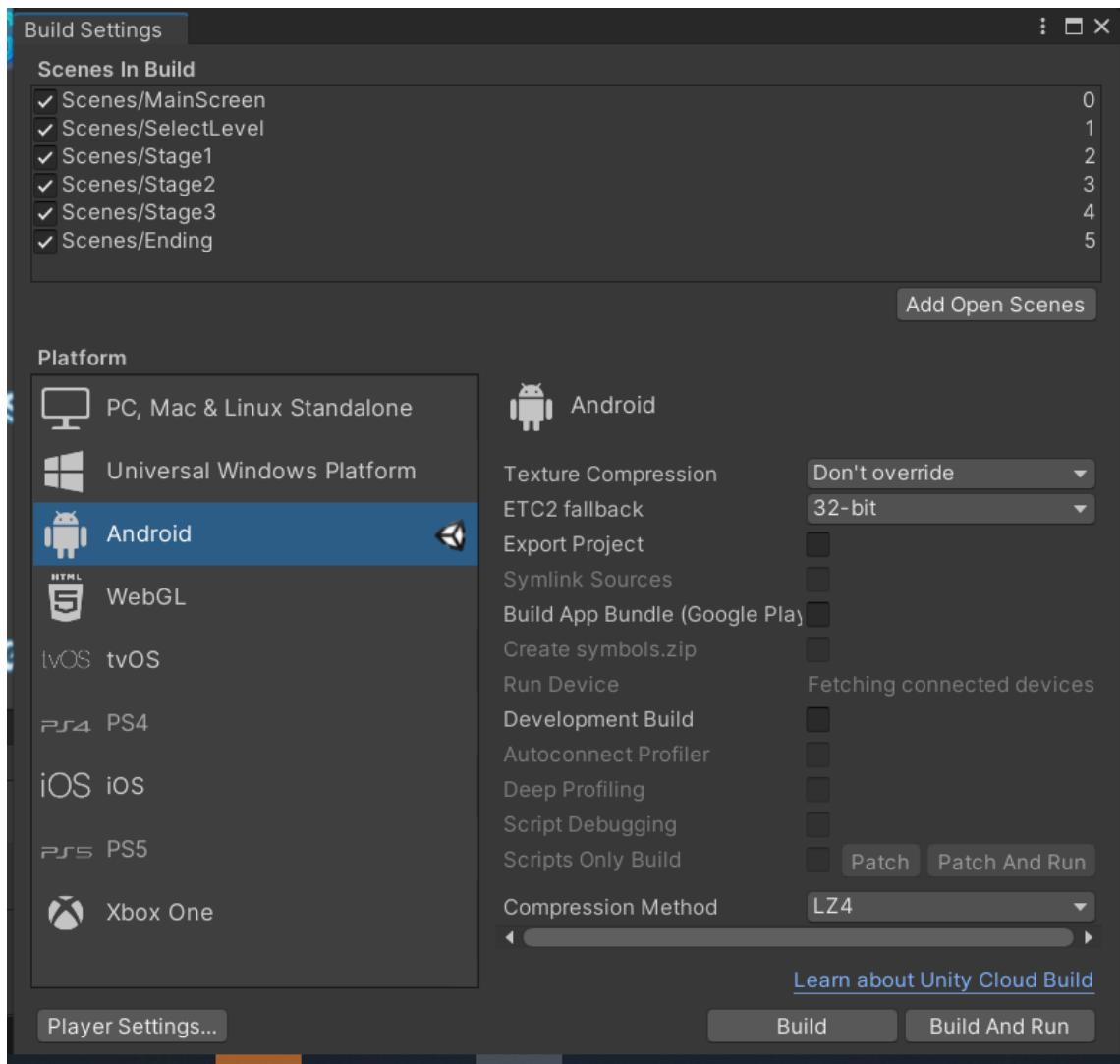
Para poder elegir a los Héroes se ha creado un panel creamos un panel y luego se han añadido un número de botones determinado por cada Héroe, es decir si son 7 personajes son 7 botones.

Tiene otro Event on Click que esta asociado a un Script por lo que a la hora de pulsar hará gestión concreta. En este caso cambiar de Skin.



Funcionamiento del Proyecto en Windows, Android y WebGL:

El procedimiento ha sido muy sencillo ya que en la Build Settings solo ha bastado con seleccionar el tipo de plataforma y configurarla.



Creación de la página web:

Para continuar con el proyecto se ha creado una página web ficticia sobre la supuesta empresa que ha desarrollado el videojuego.

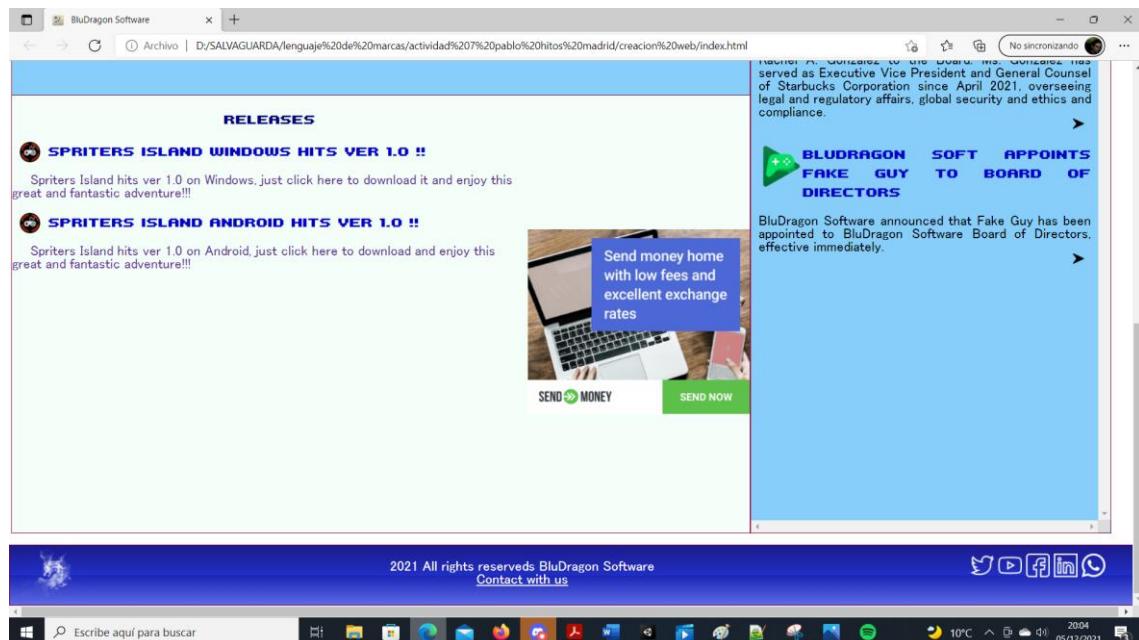
Esta página web está redactada completamente en inglés, dispone de todas las partes que por lo general suele tener una página web común y además nos permite jugar al videojuego desde su web.

Lá página web en sí está hecha con Html5 y Css3. No se ha usado ningún editor en especial para crearla, todo ha sido hecho a través de un bloc de notas.

Se ha usado Visual Studio Code con un plugin que ayuda y guía para crear páginas webs.

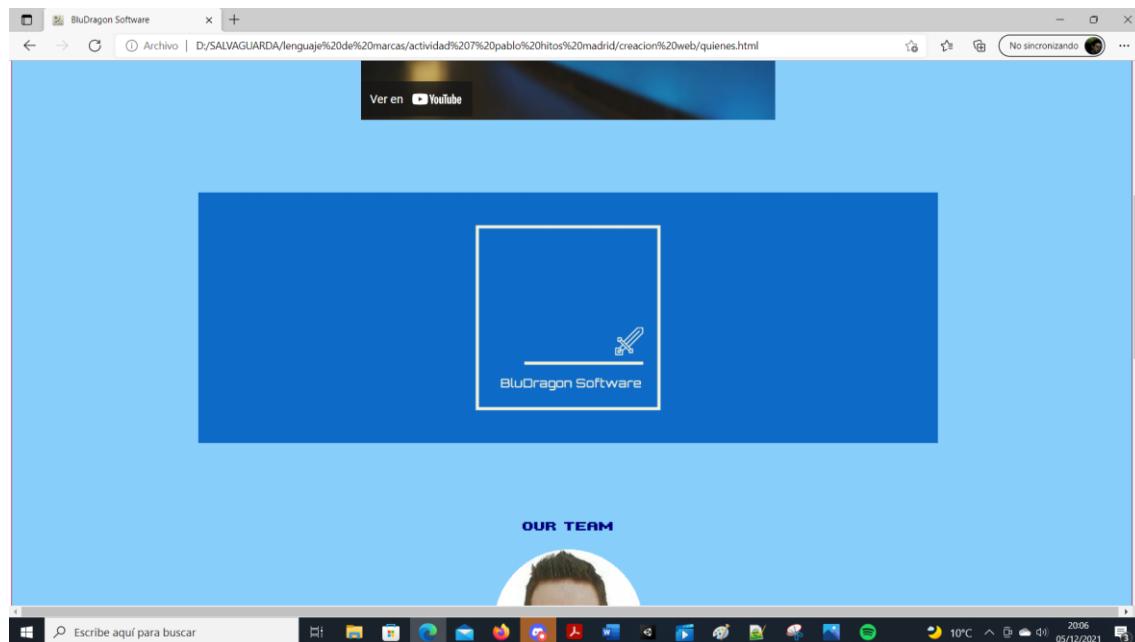
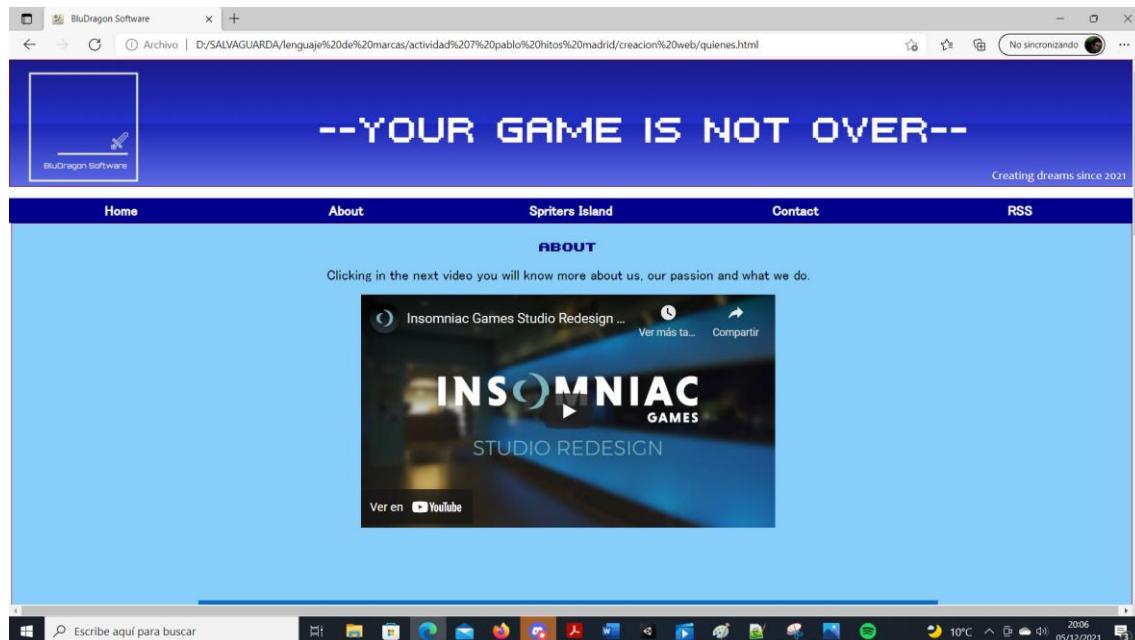
Partes de la Web:

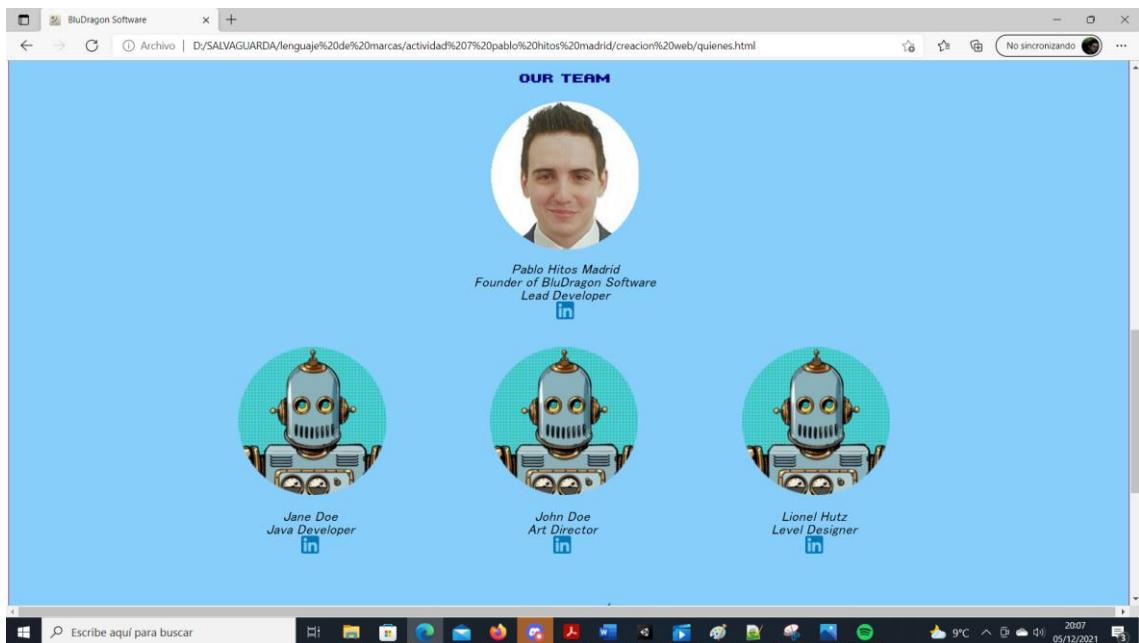
Index:



En la página principal se da algo de información sobre la empresa, en un lado hay noticias y debajo podemos comprobar la posibilidad de acceder a probar las novedades que tiene la empresa, en este caso jugar a Spriters Island.

About:





En esta parte de la web se puede encontrar información sobre la empresa a través de un video ilustrativo y además los miembros principales de esta desarrolladora indie. En cada miembro se puede ver su perfil profesional de Linkedin.

Spriters Island:

WHAT IS SPRITERS ISLAND?

Spirters Island is a side-scrolling 2D platform game. The main characters are Ninja Frog, Virtual Guy, Masked Man, Pink Man, Squirell, Tim Fox and Becca the Rabbit. These tiny heroes protect their island from the Bad Buggies, an evil gang that they want to stole all the fruits of the village and the sacred diamonds from the island.

Spirters Island is a colorful microworld, divided into several levels of various composition. In each you have some specific mission that you have achieve for if you want pass the next level for example: You have to find a sacred diamond or you have to collect all the fruits before time goes up.

The history of Sprite Islander started in 2021. Pablo Hitos the CEO of Blu DragonSoftware had to present a project for his school, so this game was born thanks to a final Project from a Higher Education Studies—The videogame was planned as the main point for the defense and presentation for the project.

The next video is a gameplay of Spriters Island.



[DOWNLOAD SPRITERS ISLAND FOR WINDOWS AND ANDROID](#)



En esta sección de la web se podrá visionar ver un gameplay del juego creado, además se podrá jugar al proyecto en su versión WebGL.

A parte existe la posibilidad de descargarlo para Android y Windows dentro de esta sección.

Contact:

Por último un formulario de contacto.

The screenshot shows a contact form titled '--YOUR GAME IS NOT OVER--'. At the top left is the Bludragon Software logo. To the right is the tagline 'Creating dreams since 2021'. The menu bar includes Home, About, Spriters Island, Contact (which is highlighted), and RSS. Below the menu is a section titled 'CONTACT WITH US' with a message: 'How can we help you? Please send us a message through this form and we will contact you as soon as possible. If you prefer another method of contact, feel free to send us an email or call us, thank you very much.' There are five input fields for Name, E-mail, Telephone Number, Subject, and a large text area for 'Text your message'. To the right, under 'OTHER WAYS TO CONTACT US', are the address 'Fake Street N°3', '18000 Granada', email 'info@bludragonsoftware.com' and 'pablo_hitsos@hotmail.com', and phone numbers '+34 XXX XX XX XX' and '+34 958 00 00 00'. A small icon of an envelope with a lock is shown.

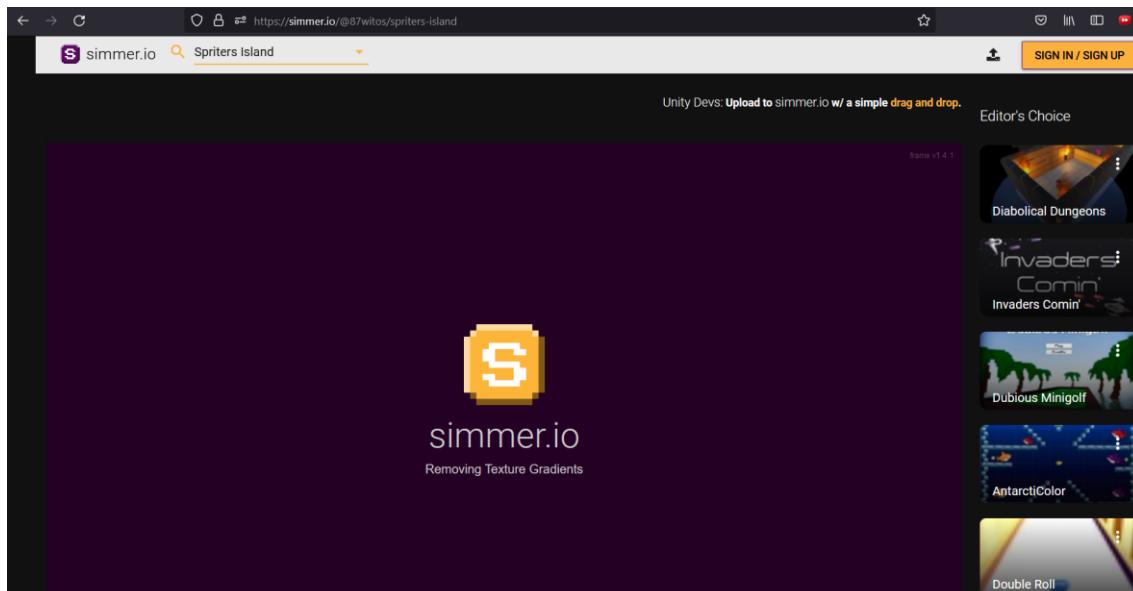
Auto Instalador: Se ha usado Ino Setup

Simmer.io y Itch.io

El juego se ha subido a estas dos plataformas y es posible jugar en ellas.

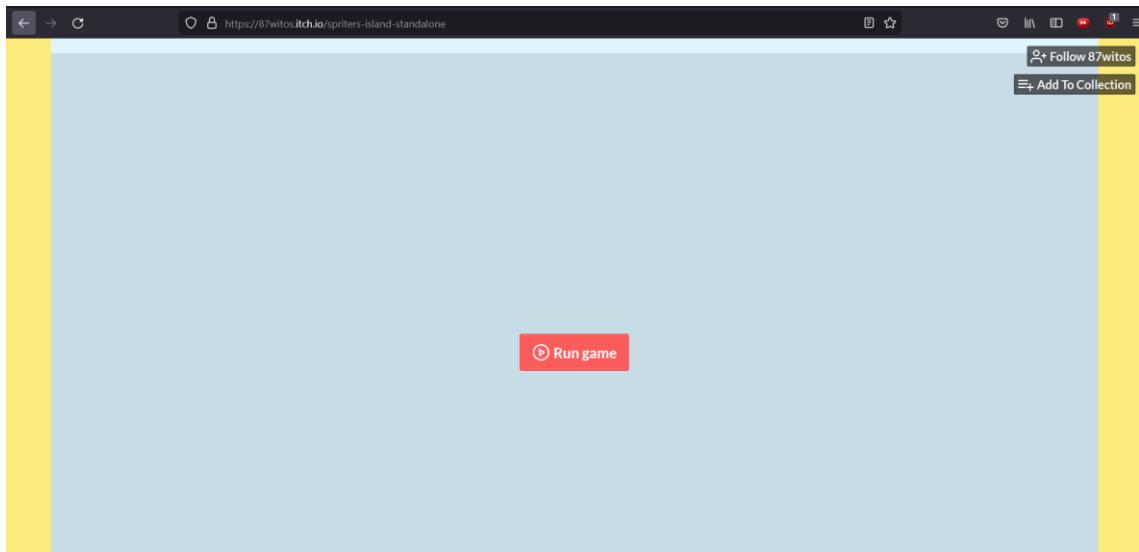
Simmer.io

<https://simmer.io/@87witos/spriters-island>



Itch.io

<https://87witos.itch.io/spritters-island-standalone>

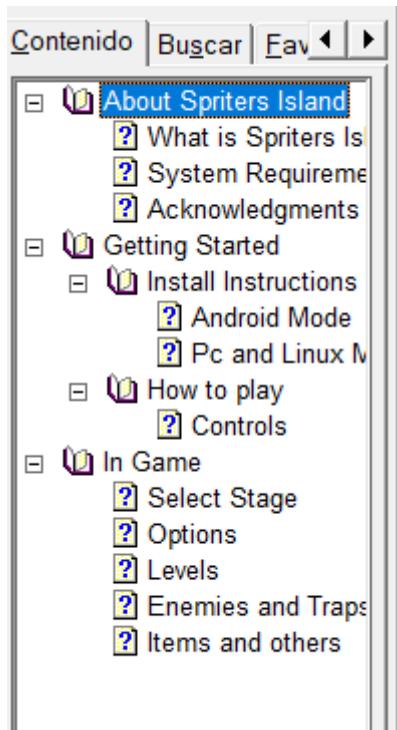
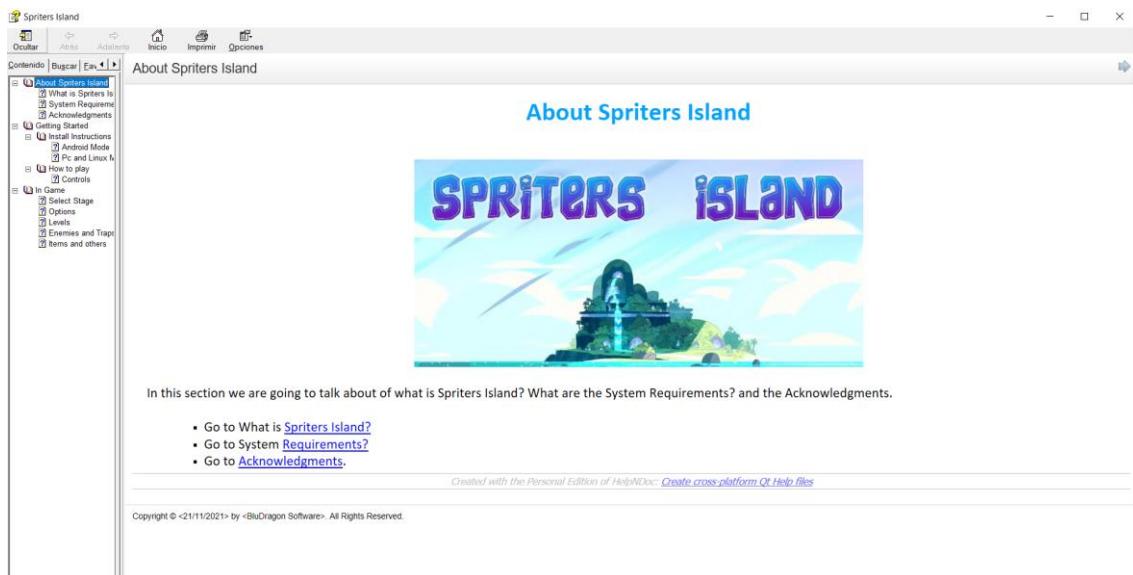


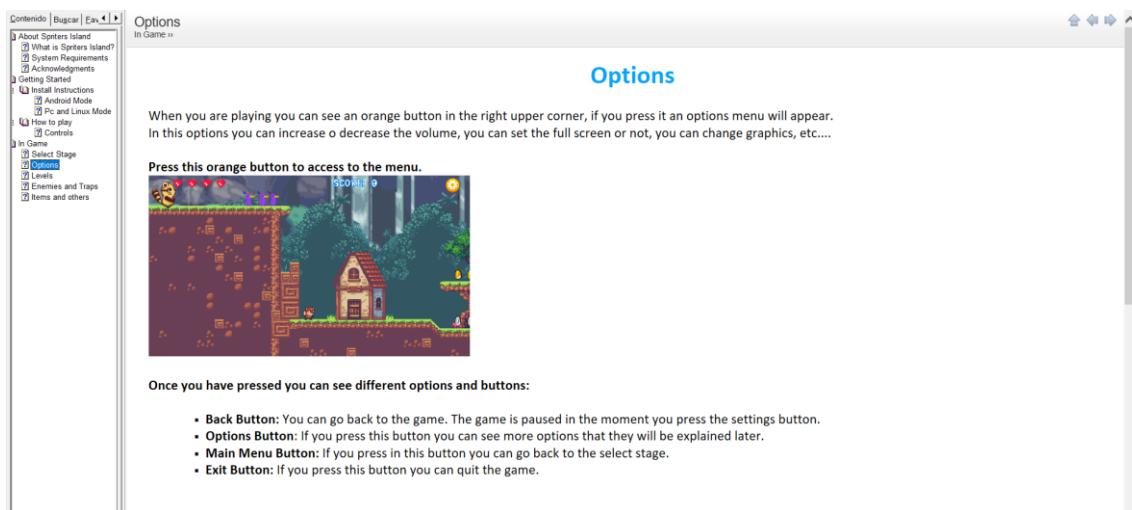
Manual de Instrucciones:

A través del programa Help n Doc se ha hecho un manual de instrucciones totalmente en inglés sobre el juego. En este se exponen todos los detalles de este proyecto, es decir: Información de su creación, como instalarlo, plataformas disponibles, tipos de enemigos, accesorios, niveles, etc....

Para poder ver manual completo es recomendable su descarga a través del git correspondiente.

[87witos/Spritters-Island-PC \(github.com\)](https://github.com/87witos/Spritters-Island-PC)



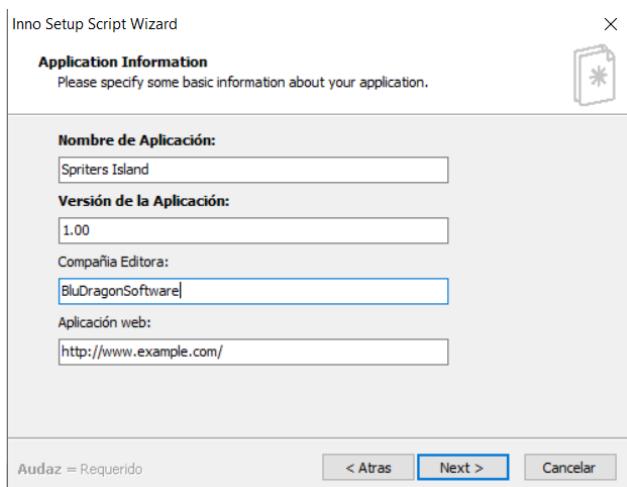


Instalador:

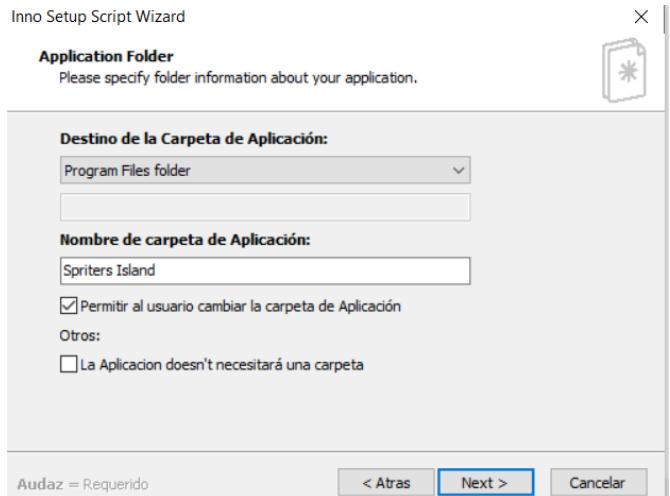
Para crear el instalador se ha usado el programa Inno Setup.

Los pasos fueron los siguientes:

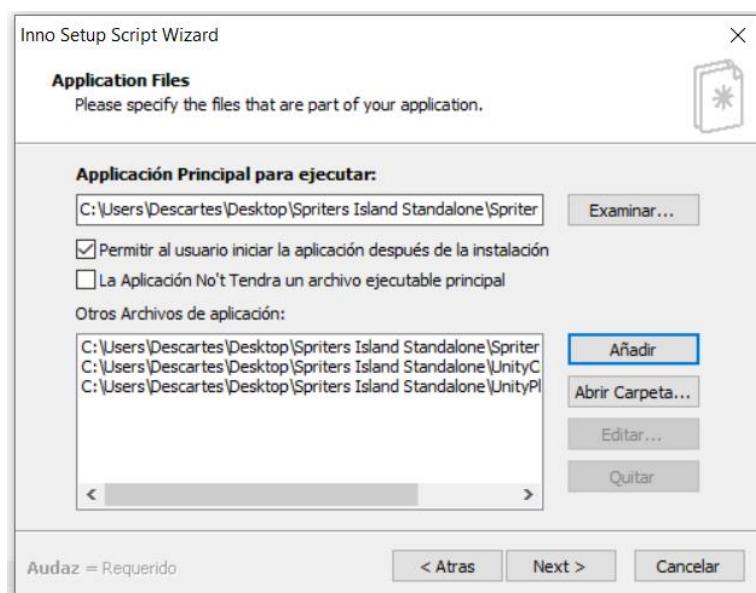
Se creó un nuevo proyecto con el nombre del juego.



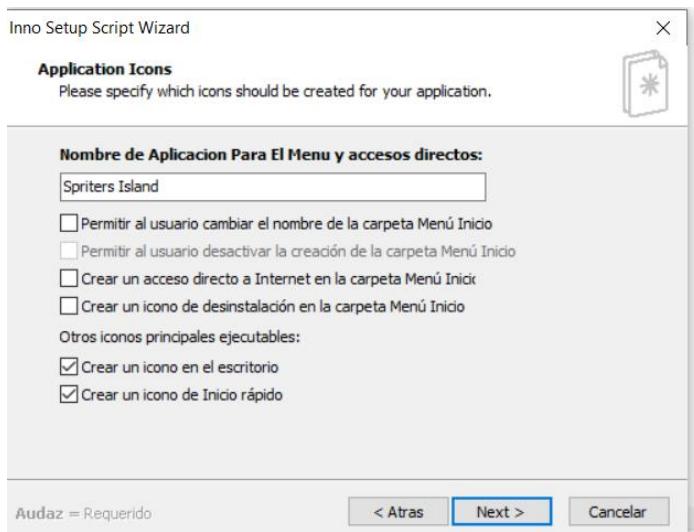
Pulsamos siguiente



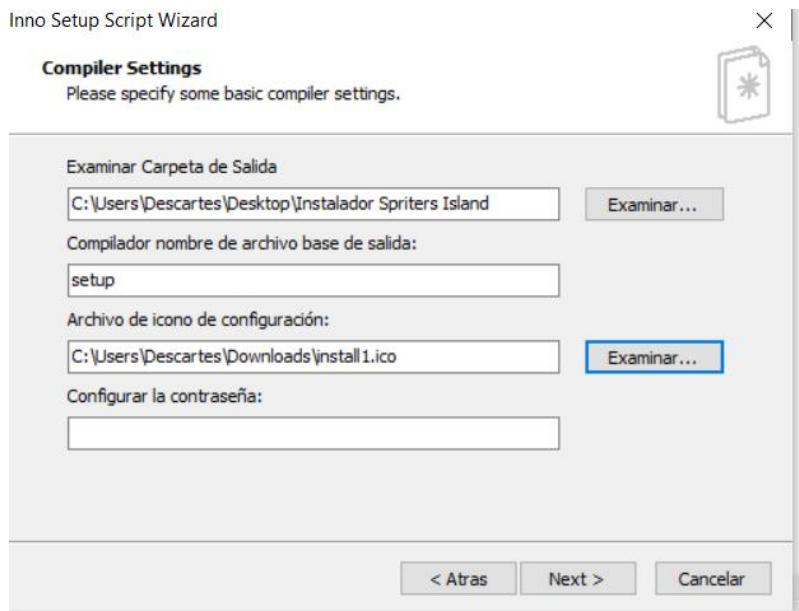
Se selecciona el exe del juego y se incluyen el resto de archivos



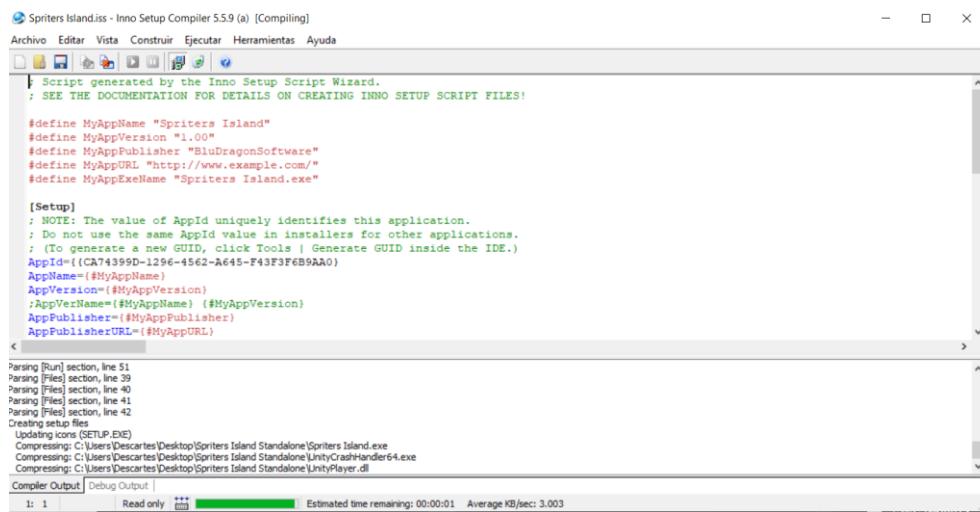
Se deja la configuración de la siguiente manera:



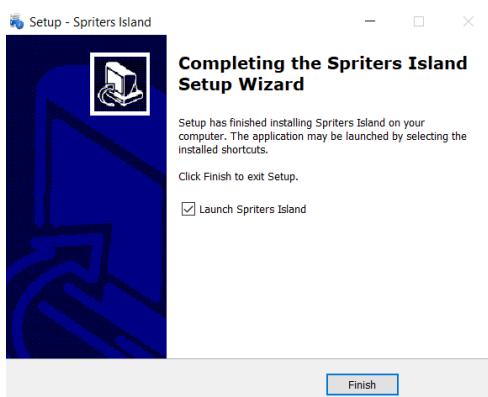
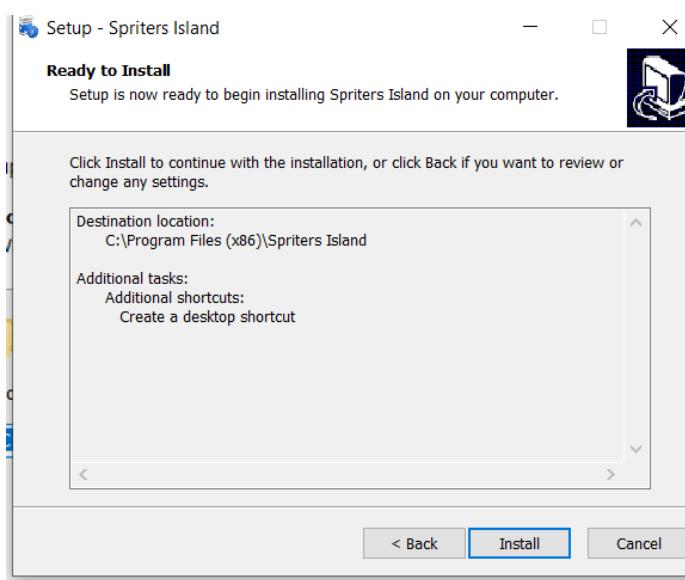
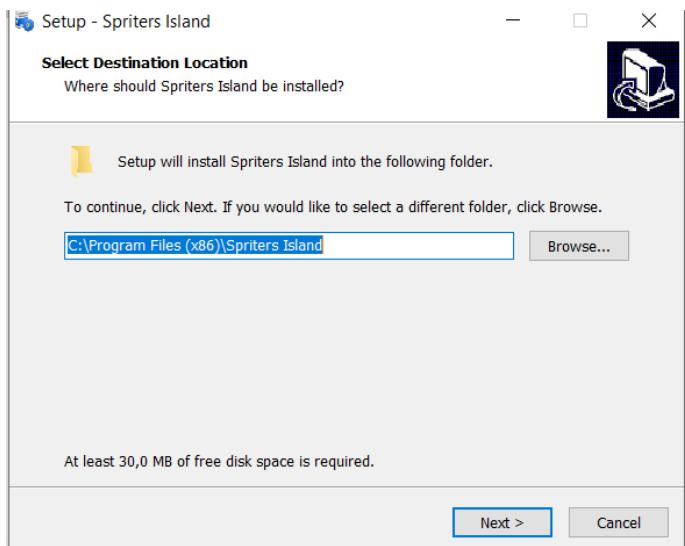
Se termina de configurar



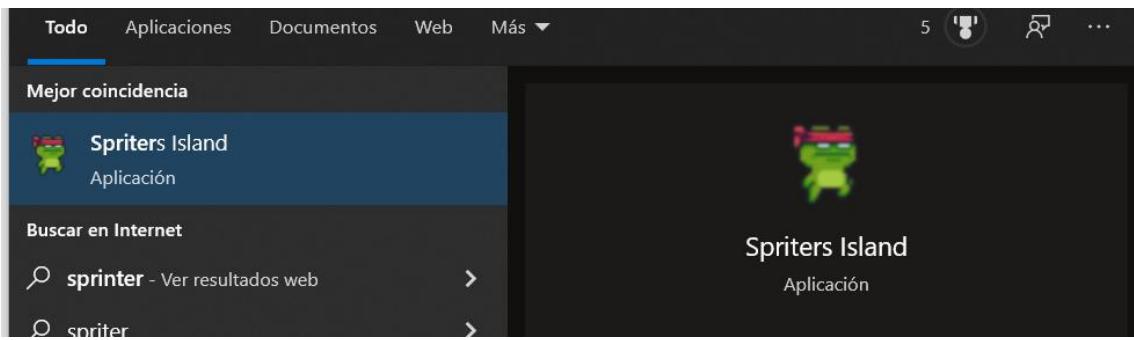
Se compila el proyecto



Aquí se puede ver su funcionamiento e instalación



Aquí se puede ver instalada correctamente



APARTADO RELATIVO AL GDD INCLUIDO EN ESTE PUNTO:

1. Título

- Título con fuentes de letra sugerente, logo o gráfico.

La fuente utilizada es la de Los Goonies



- Información de contacto.

BluDragon Software

Email: pablo_hits@hotmail.com

Whatsapp: +34 651 50 40 21

Git: <https://github.com/pablohitos>

- Plataformas objetivo.

El juego estará diseñado para ser jugado en:

1. Android (Versión Mínima 4.0)
2. Windows (Windows 10 recomendado)
3. Navegador de Internet.

- Edad del Target (público objetivo): El target es de 8 a 70 años.

- **PEGI:** Tendrá calificación de PEGI 3. El contenido de los juegos con clasificación PEGI 3 se considera adecuado para todos los grupos de edad. El juego no debe contener sonidos o imágenes que puedan asustar a los niños pequeños. Se acepta una forma de violencia muy leve (en un contexto cómico o en un entorno infantil). No debe haber palabras malsonantes.
- **Fecha planificada de lanzamiento:** La fecha planificada de lanzamiento es el 10 de Diciembre de 2021

2. Presentación del juego

- **Resumen de la historia del juego (tres párrafos), describiendo donde tiene lugar, los personajes y el conflicto:**

Comienzo: La Isla Spriters es un lugar pacífico y protegido gracias al diamante sagrado cuyo origen es desconocido. En esta isla conviven nuestros héroes con el resto de aldeanos. Los habitantes de esta isla se alimentan de frutas especiales que recolectan durante todo el año.

Una noche, la pandilla de malhechores conocidos como los Bad Buggies (Bat, Blue Bird, Plant “The Gun”, Green Dragon, Rebel Pig, Bee, Zombee, etc..), se adentraron en la pacífica isla para robar el diamante sagrado y de paso llevarse toda la comida de los aldeanos, es decir, las frutas mágicas. Nuestros héroes: Ninja Frog, Virtual Guy, Masked Man, Pink Man, Squirell, Tim Fox and Becca the Rabbit, se encargarán de recuperar el diamante sagrado y además todas las frutas mágicas que estos villanos han ido robando.

Desenlace: Nuestros héroes pasarán por distintas fases, una vez recojan las frutas y recuperen el diamante sagrado, toda la isla volverá a la normalidad y nuestros héroes podrán disfrutar de un bonito atardecer mientras todo el mundo les agradece el esfuerzo de haber recuperado la paz en la bonita isla.

- **Directrices del Juego (todo lo que pueda describir para entender el juego y la experiencia atener), describe brevemente el flujo de la acción del juego en el contexto de los lugares en los que se encontrará el jugador:** Spriters Island es un juego en scroll horizontal al igual que juegos que marcaron un antes y un después como Super Mario Bros, Donkey Kong Country, etc...

Los personajes se moverán en un entorno 2D en el que tendrán que ir salvando obstáculos de todo tipo (precipicios, pinchos, bolas de pinchos que se mueven entre 180 o 360 grados, cubos de pinchos que se mueven de arriba abajo, lava, etc....), además de encontrarse este tipo de trampas todas las pantallas estarán plagadas de enemigos de distinta clase y dependerán de la fase en la que nos encontremos. Se podrán encontrar desde enemigos voladores como los Blue Bird, Bat o al temible Zombee o se podrán encontrar enemigos terrestres como los Mushrooms (setas que van de un lado a otro), los Rebel Pigs (enemigos que si saltas encima de ellos su velocidad aumenta). Básicamente por la descripción se deberá tener en cuenta que se trata de un juego en el que hay que tener reflejos y medir cosas como los saltos, los tiempos y aprovechar los power ups de las pantallas.

El jugador tendrá que enfrentarse a estos enemigos y estos obstáculos en distintas partes de Isla con una temática muy diferente en cada nivel, es decir, se irán explorando la isla conforme avancemos.

- **Personaje:** Son 7 personajes seleccionables que todos tienen las mismas habilidades, tienen una velocidad similar, un número de vidas similar y un salto similar, sus nombres son: Ninja Frog, Virtual Guy, Masked Man, Pink Man, Squirell, Tim Fox and Becca the Rabbit
- **Cámara:** La cámara es en tercera persona por lo que se podrá ver además de los enemigos, al héroe y su aspecto o como es dañado.
- **Estilo de juego:** El estilo de juego es un Plataformas 2D estilo “old School”, como Super Mario Bros o Donkey Kong Country.
- **Ambiente:** Se desarrolla en una Isla con diferentes ambientes (selvático, bosque y minas antiguas). Es un ambiente natural.
- **Objetivos:** Conseguir todas las frutas de los niveles para poder pasarlos y conseguir el

diamante sagrado. Un objetivo secundario es conseguir el mayor número de monedas posible para aumentar el score y vencer el mayor número de enemigos posible para aumentar dicho score.

- **Enemigos:** Hay varias clases de enemigos, los terrestres que van de un sitio a otro o lanzan balas, los voladores que pueden estar quietos en el aire o ir desde 2, 3 y 4 puntos diferentes cada uno a distinta velocidad, enemigos voladores que lanzan artefactos y enemigos que persiguen como fantasmas. Aparte de todo esto hay trampas como pinchos, bolas de pinchos, cajas de pinchos, precipicios etc...
- **Fases:** De momento son 3 fases cada una con una temática distinta. Una pequeña jungla o Selva bajo cascadas, una mina de piedras preciosas dentro de un bosque y una pantalla que se desarrolla en las alturas y en lo mas profundo de un bosque.

3. El Player:

- Descripción del Héroe que se va a jugar, edad, sexo, antecedentes.

Los héroes:

Ninja Frog: Principal protagonista. Una rana ninja entrenada por el maestro mas temible del ninjutsu con su aspecto imponente los enemigos ya tiemblan al verlo. Vive en la Isla Spriters y tiene 14 años.

Virtual Guy: Enamorado de la tecnología, a veces no sabe si vive en el mundo real o en un mundo virtual. Su seña de identidad son sus Gafas VR. Su sueño es ser el mejor jugador de videojuegos. Vive en la Isla Spriters y tiene 13 años.

Masked Man: Sabio chamán y héroe. Nadie ha visto su cara jamás ya que la lleva tapada con una mascará Zulu. Cuando un habitante de la isla esta en peligro no lo piensa dos veces, ya sea para curarlo o ayudarle. Tiene 104 años y sabe algo sobre el cristal sagrado que no cuenta.

Pink Man: El rosa no es su color, es su forma de vida, es muy ágil y no lo duda dos veces en adentrarse en una aventura junto con Ninja Frog. Tiene 14 años y fue a la escuela con Frog.

Squirell: Esta ardilla habitaba en lo alto de los árboles, tras ver lo sucedido se unió al grupo de héroes y su ayuda es inestimable. No soporta que hayan robado las frutas. Tiene 3 años.

Tim Fox: Pequeño zorro que con corazón de aventurero cuyo sueño es ser como Pinkman y Frog. Tiene 5 años y aún va a la escuela.

Becca the Rabbit: Becca es una coneja hiperactiva que cuando el deber llama ahí está ella dispuesta a echar una mano, ya sea en un concurso de tartas o recuperar el tesoro de la Isla. Es muy coqueta y tiene 6 años.

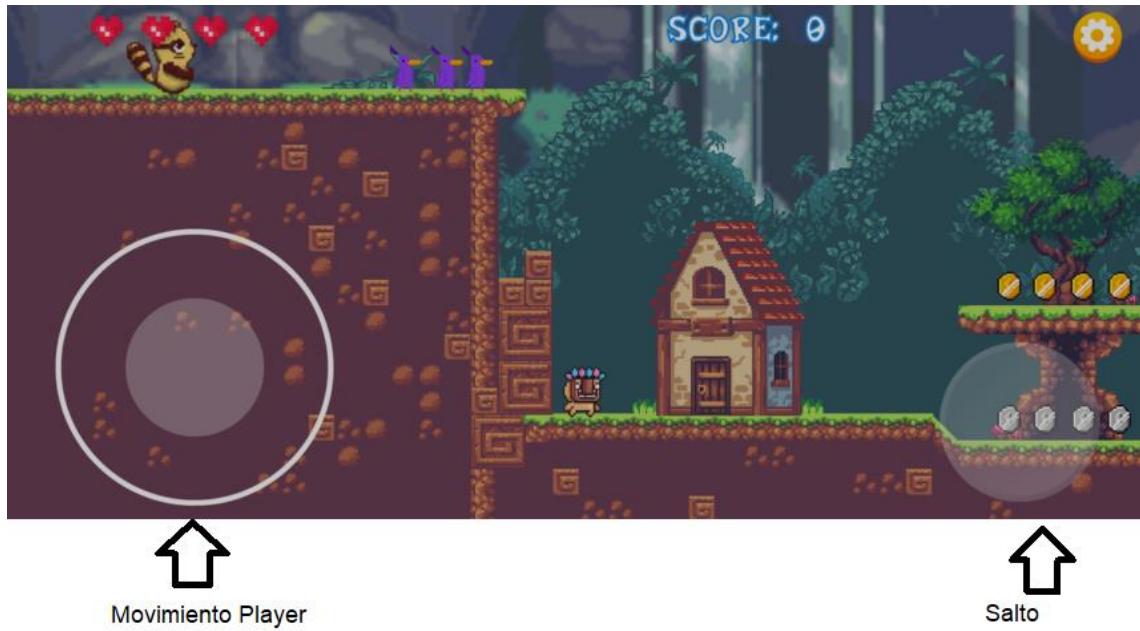
▪ **Qué habilidades tiene. Arte conceptual. Armas.** Las habilidades de estos héroes son principalmente físicas. Es decir vencen a los enemigos saltándoles encima. No usan armas como tal aunque se apoyan de power ups en forma de gemas que les pueden dar velocidad o aumentar su salto durante 10 segundos.

▪ **Mapeado básico de control usando un mando de juego, como si fuera el manual de uso o de la interfaz de interactividad si fuera un móvil o Tablet**

Mapeado PC:



Mapeado Android:



4. Jugabilidad

- Cómo se aplica el estilo y género del juego. Secuencia de juego. Secuencia de experiencia.

El juego dispone de momento de 3 niveles. Cada nivel tiene una misión distinta.

Nivel 1: Búsqueda de el diamante sagrado aunque también si se quiere obtener un score mayor es necesario eliminar a todos los enemigos posibles y coger todas las monedas que se puedan.

Nivel 2: Es obligatorio coger todas las frutas que se encuentran en el nivel para poder pasarlo. Hay que eliminar a todos los enemigos posibles y coger monedas para aumentar tu score.

Nivel 3: Necesario coger todas las frutas que se encuentran en el nivel antes de que el tiempo se agote. Si no el nivel de reinicia, es posible eliminar enemigos y coger monedas para aumentar score.

- Describir minijuegos y mecánicas. Usar diagramas. El juego no dispone de minijuegos dado a que en ningún momento se estableció ni se pactó que iba a tenerlos además de que no entraba dentro del tiempo que tenía fijado para programar el proyecto.

- **Enumerar detalles relativos a la plataforma a la que va destinada, disco duro, tarjeta de memoria, pantalla táctil, multiplayer con pantalla dividida, si usa la cámara, uso de internet, servidores, compras en aplicación y cualquier cosa necesaria para entender el alcance de la complejidad del proyecto.**

El juego va destinado a Plataformas Windows y se podrá descargar desde github e itch.io, aparte el juego podrá jugarse en los navegadores desde itch.io, simmer.io y la propia web de BluDragon Software.

El juego se podrá jugar también en Smartphone a través de una APK diseñada.

Requisitos PC:

- Es necesario disco duro para jugar y necesitamos al menos 100 megas.
- En cuanto a tarjeta de memoria no es un juego con unos recursos elevados por lo que es valida incluso la tarjeta integrada de un portátil. No requiere grandes requisitos al ser un juego en 2d.
- En pc no es necesario pantalla táctil, usamos ratón para manejarnos junto con el teclado.
- No es necesario estar conectado a Internet.

Requisitos Android:

- 100 megas de instalación.
- Apto para móviles de gama baja.
- Mínimo 1GB de ram.
- Android 4.0 como mínimo
- Uso de la pantalla tactil para manejar el panel de mandos táctil.
- No es necesario estar conectado a Internet.

Requisitos Navegador:

- Navegador Firefox, Google Chrome o Edge actualizado en última versión.
- Mínimo 2gb de Ram.
- 100 megas de espacio libre.
- Necesario uso de internet para poder jugar.

5.Mundo del juego

- **El mundo en el que se desarrolla, y sus sub ambientes. Descripciones cortas.**
Dónde tiene lugar el gameplay.

El juego se desarrolla en un mundo ficticio, concretamente en una isla con personajes humanoides. Dentro de esta isla tenemos al menos tres subambientes:

Nivel 1: El nivel 1 representa una pequeña Selva donde se podran encontrar enormes cascadas. Es un ambiente relajante.

Nivel 2: El player comenzará en una mina de extracción de piedras preciosas para posteriormente acabar en un bosque con aspecto otoñal.

Nivel 3: El player comenzará en el interior de un bosque. Esta pantalla se desarrolla en lo alto de los arboles para acabar finalmente en un lago con montañas nevadas.

- **Qué hay en cada lugar para el Personaje:**

Nivel 1: Hay un diamante sagrado y diferentes monedas, además de enemigos y trampas y una pequeña mazmorra y un valle con lava.

Nivel 2: Frutas mágicas, monedas, gemas con power up que aumenta el salto, diferentes monedas, además de enemigos y trampas.

Nivel 3: Frutas mágicas con tiempo límite y diferentes monedas, además de enemigos y trampas y gemas con power up que aumenta la velocidad.

- **Que personalidad tiene cada sitio.**

Nivel 1: Naturaleza pero con carácter relajante y colorido, con música acorde al espacio al que envuelve al personaje.

Nivel 2: Interior de una mina de piedras preciosas, posteriormente bosque de carácter otoñal, colores menos vivos pero relajante junto con música acorde a la mina.

Nivel 3: Naturaleza, bosque colorido con carácter relajante.

- Que música lo va a ambientar.

Pantalla de Inicio:

<https://www.youtube.com/watch?v=IrtTu89e98M>

Selección de nivel:

<https://www.youtube.com/watch?v=HaWaRAaXYug>

Nivel 1 Jungla:

<https://www.youtube.com/watch?v=skxs1efVG98>

Nivel 2 Mina:

https://www.youtube.com/watch?v=Hijff_04nps

Nivel 3:

<https://www.youtube.com/watch?v=qIk6YFTzckc>

Créditos:

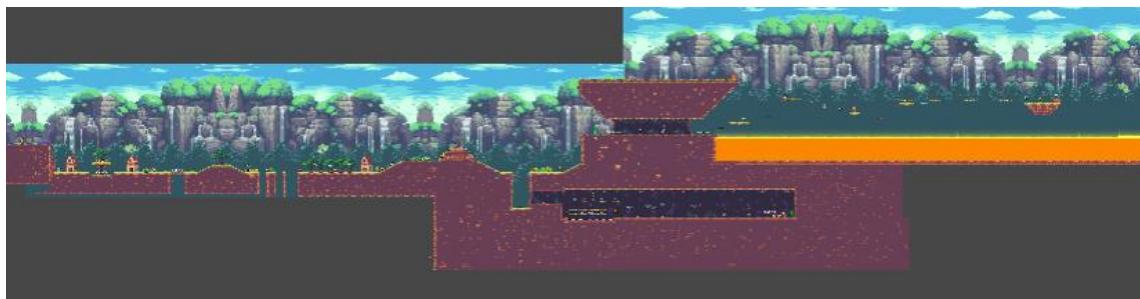
<https://www.youtube.com/watch?v=3vXHR3nvoI8>

- Como están conectadas las localizaciones entre si.

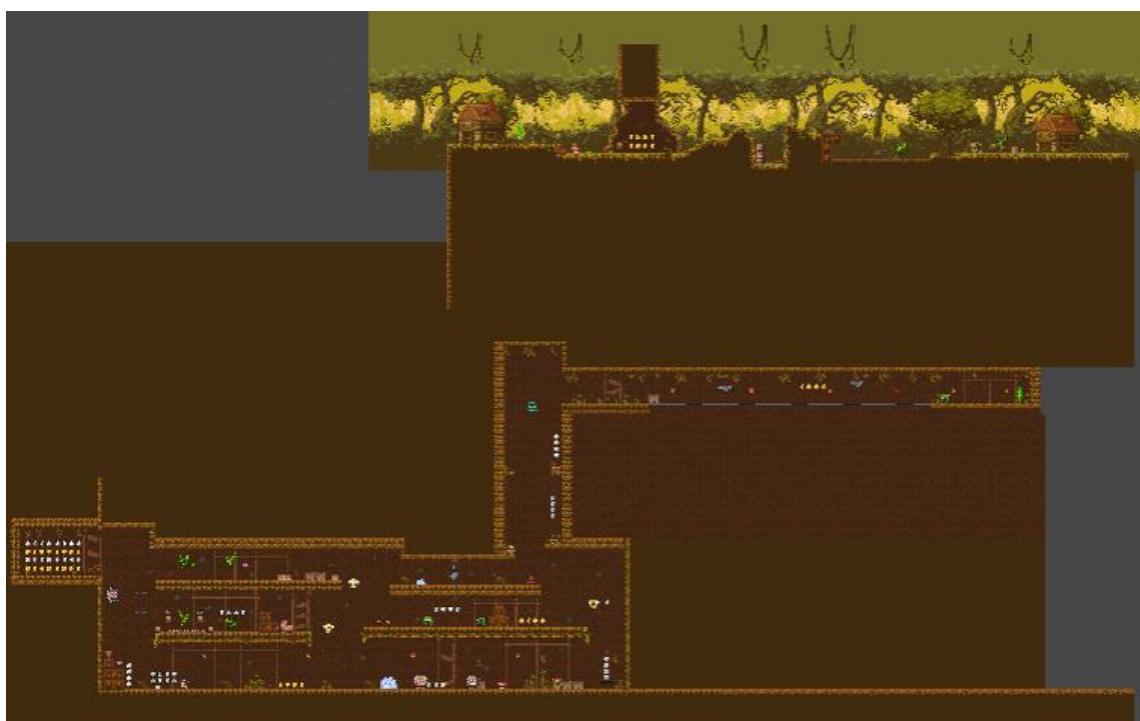
Las localizaciones entre sí no están conectadas realmente, cuando se completa nivel el player va a otro y así sucesivamente. El personaje va rodeando la isla y visita diferentes rincones.

- **Incluir algún mapa o diagrama de flujo para entender como el personaje se mueve por el mundo.**

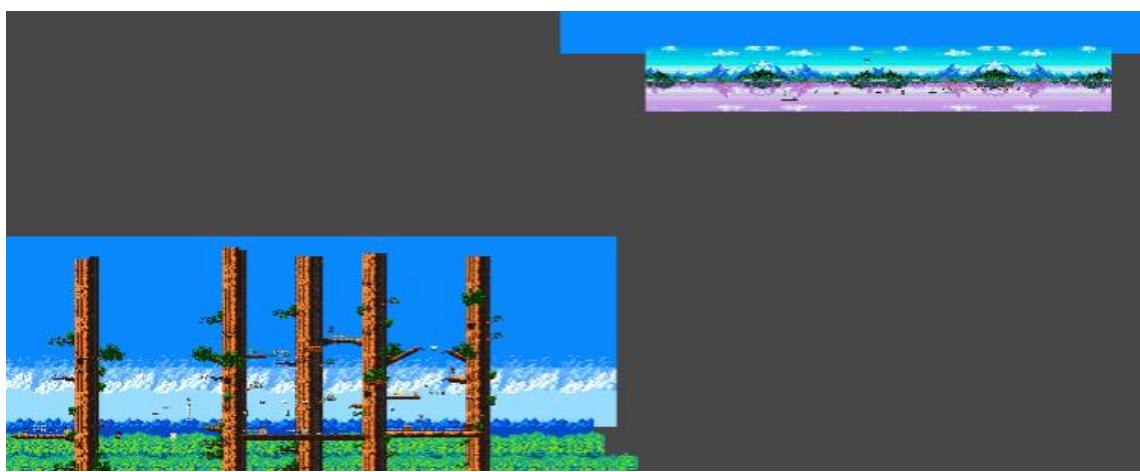
Nivel 1:



Nivel 2:



Nivel 3:



6.Experiencia de juego / Interfaz

▪ **Todo lo que sirva para describir las sensaciones que debe producir el juego.** El juego quiere transmitir relajación a través de su música, es una música ambiente que te ayuda a introducirte en el nivel, a su vez los niveles también están diseñados para generarte sensación de tensión o alerta ya que muchos de ellos no son fáciles y pueden enfadar al jugador si no tiene la suficiente destreza. También intenta desprender un aire nostálgico a través de un estilo pixel art y unos sonidos de recogidas de monedas, etc... que te harán recordar a juegos de la old school.

▪ **Qué se ve primero en el juego. Que emociones debe suscitar el juego al completo.** En el juego se ve una pantalla de inicio en la que indique pulsa aquí para continuar. En esa escena se podrá observar ver la isla del videojuego en su totalidad, es decir, toda la aventura del juego transcurrirá en esa isla. Aparte se ve el título del video juego.

▪ **Como la música o el sonido acompaña estas emociones.** Se ha elegido muy cuidadosamente la música y se ha escogido música que vaya acorde a las escenas y generen sensación de calma y tranquilidad o incluso nostalgia y que sean canciones muy melódicas. Casi todas las canciones son de David Wise, famoso compositor de grandes obras como Donkey Kong Country de Super Nintendo.

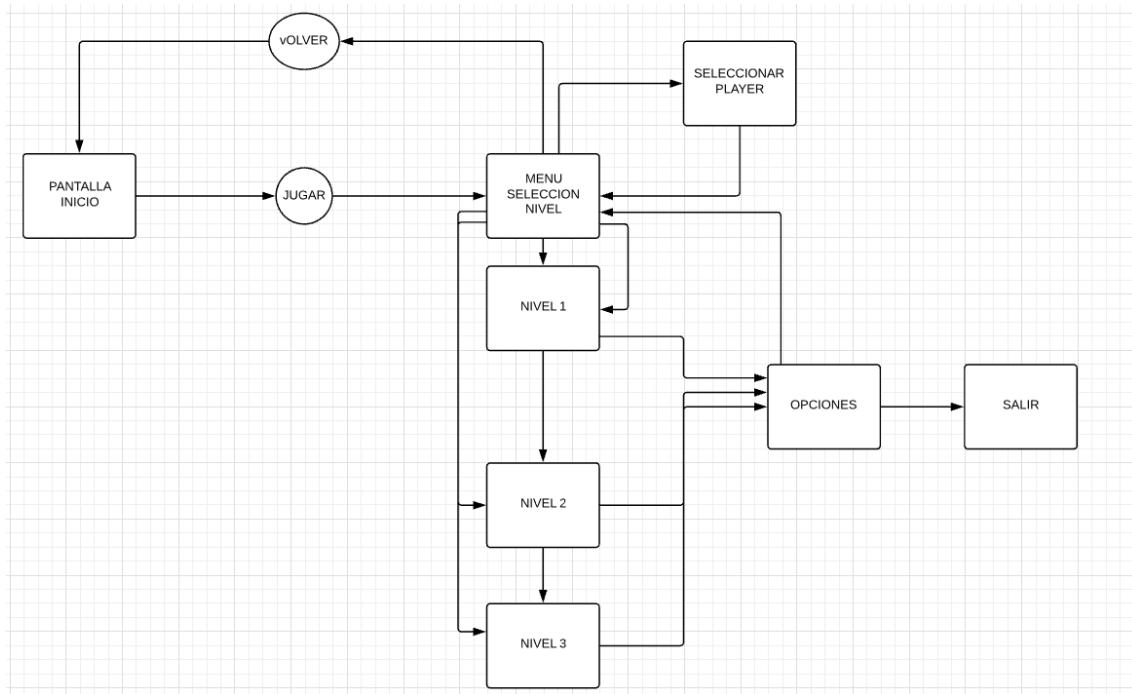
Había intención de incluir otro tipo de canciones como estas pero no llegaban a encajar con el nivel o temática del juego:

<https://www.youtube.com/watch?v=bWa6gVunVLE> Hot Head Bop [Donkey Kong Country 2]

<https://www.youtube.com/watch?v=dUHxRQ4ROn4> Donkey Kong Country - Life in the Mines [Restored]

▪ **Qué otros mecanismos hay que definan la experiencia que se busca.** Se busca cierta sensación de tensión poniendo niveles complicados ya sea con enemigos inesperado u obstáculos complicados, por último se busca sensación de nostalgia a través de un estilo pixel art y sonidos que recuerdan a juegos de la “Old School”.

- Diagrama de flujo de uso de navegación de la interfaz.



7.Mecánicas de juego y Power-ups

- **Qué mecánicas debe seguir el jugador para avanzar en el juego.** Las Mecánicas que ha de seguir el jugador para avanzar en el juego son simples. Ha de vencer a los enemigos a base de saltos, esquivar trampas y precipicios y según el nivel cumplir unos objetivos. En el primer nivel conseguir un diamante, en el segundo nivel recoger todas las frutas y en el tercer nivel recoger todas las frutas en un límite de tiempo.

El juego tiene power ups como aumento de velocidad o aumento En el salto.

- **Qué peligros o retos debe superar (mecánicas que dañan al jugador).** Debe superar a una gran cantidad y variada de enemigos que cambian según el nivel además de obstáculos y precipicios.

- **Qué power-ups va a encontrar disponibles y como afectan.** Esta el power up de aumento de velocidad que aumenta la rapidez del personaje y el aumento del salto del player que hace salte mas alto. Estos power ups duran 10 segundos.

- **Qué cosas puede colecciónar.** Puede coleccionar frutas, un diamante, gemas de power up y monedas plata y oro.
- **Qué sistema de economía/compras sigue la mecánica si tiene alguno.** No tiene un sistema de economía.

8. Enemigos y Bosses

- **Mecánicas y AI de los jefes de fase o encuentros diferenciadores .** Los enemigos tienen distintos tipos de AI. Una AI que te persigue alrededor de toda la pantalla. Otra AI que se mueve a diversos puntos de la pantalla previamente indicados, otra AI que lanza proyectiles según los segundos estipulados y otra AI con RayCast que una vez te alcanza de lanza un proyectil.

No hay jefes como tal solo enemigos en la pantalla.

- **Qué enemigos o peligros se encuentra el jugador y como le afecta que los pase o que le derrote.** Se encuentra con enemigos terrestres que van de un punto a otro, enemigos voladores que pueden ir a distintos puntos, enemigos que lanzan artefactos, enemigos que una vez pasas por su rango te lanzan artefactos y enemigos que te pueden perseguir por toda la pantalla. No afecta en nada que los derrote, la misión principal del juego es otra.

- **¿En qué entornos aparecen?** Aparecen en todos los niveles.

- **¿Cómo los vence el jugador?** Saltando sobre su cabeza.

- **¿Qué obtiene el jugador por derrotarlos?** Obtiene puntos para su score.

9. Animaciones

- Videos, cinemáticas o escenas fin de juego o entre niveles, y su estilo.

El juego tiene una especie de cortinilla que una vez pasas una fase se activa y luego aparece un título que dice fase completada.

A parte una vez finalizados los niveles podemos ver una fase de escena de fin del juego que se le da la gracias al player por jugar.

El estilo de todo lo descrito anteriormente va acorde a todo el estilo del juego en sí, ya que usa la misma fuente principal, colores, etc...

- [¿Cómo se presentarán al jugador?](#)

Se presentan finalizando los niveles.
▪ trailer

<https://www.youtube.com/watch?v=497m9oXGv8I>

Material extra

- [Posible material descargable, ¿qué material podrá desbloquear el jugador?](#)

No tiene material descargable, en cuanto a aspectos desbloqueable lo mas destacable son los niveles que son jugables conforme se completa el anterior.

- [Extras en multiplayer](#) No tiene multiplayer

- [¿Qué incentivos hay para que el jugador juegue de nuevo?](#) El incentivo sería poner un cronometro para hacer time attacks y ver quien es mas rápido.

- [¿Qué otros juegos serán tu competencia cuando lances el videojuego al mercado?](#) Super Mario Bros, Donkey Kong Country, Super Meat Boy, etc.....

Finalización de GDD.

11. Conclusiones y mejoras del proyecto, resumen de los objetivos conseguidos, así como de los resultados obtenidos si proceden.

En cuanto a conclusiones se debe indicar que en un plano general y a nivel personal estoy satisfecho con el trabajo realizado, es un proyecto al que le he dedicado bastantes horas y puesto mucha ilusión.

Dicho proyecto se comenzó por Julio ya que se quería introducir el mayor número de cosas posibles y presentar un producto o un proyecto que fuese medianamente aceptable y resultara del agrado de los usuarios.

Se ha querido que este proyecto sea la experiencia más cercana a lo que supone comprar un videojuego de verdad y ya finalizado. No se quería que fuese un juego donde directamente el personaje apareciera en escena dando saltos, etc... se quería que tuviese un mínimo de profesionalidad, que presentara algo de dinamismo y que cada fase fuese un reto diferente.

Otro punto a destacar y que por lo general a muchos programadores se les escapa, es que a título personal estoy bastante satisfecho con la estética final del juego, se ha procurado cuidarla al máximo posible dentro de las capacidades actuales como programador, aunque como todo, siempre puede ser mejorable. Todo esto no quita que el diseño de los niveles en el fondo hayan sido muy elaborados y se ha intentado introducirles la mayoría de detalles posibles, no se quería que fuesen sprites puestos al azar. Otro punto importante es el tema del menú, se intentó que tuviese una estética casual como algunos juegos actuales y finalmente la música, que fue elegida expresamente y de forma concienzuda.

Mejoras del proyecto:

Una vez finalizada la entrega del proyecto me gustaría seguir trabajando en él. Estas son las cosas que me gustaría mejorar :

- Arreglar tema de colisiones con las “bullet o balas”, es decir, una vez que colisionen con el player se destruyen, ya que ahora mismo se destruyen pasadas unos tres segundos.
- Añadir una inmunidad al player de unos 3 segundos cuando le quitan vida o colisiona con un enemigo.
- Cambiar el joystick de Android por una cruceta (lleva Joystick porque era el asset gratuito).
- Añadir un Highscore.
- Añadir particulas e iluminación, a personajes enemigos y ambiente.
- Añadir soporte para mandos.
- Compatibilidad en videoconsolas.
- Añadir multiplayer.
- Probar nuevos estilos en la cámara.
- Añadir disparos en el player.
- Añadir mas stages.

Resumen de los objetivos conseguidos:

Todos los objetivos que se propusieron en el anteproyecto se han conseguido ya que se dijo que se haría un juego en 2D de plataformas completamente en inglés en el que habría niveles seleccionables y que se irían desbloqueando, personajes seleccionables, pantallas con diferentes temáticas, objetivos distintos en cada nivel, recogida de fruta, monedas, power ups, unos 10 tipos de enemigos, varios tipos de trampas, etc...

También se dijo que este proyecto funcionaría en ordenadores, en smartphones y se podría jugar en un navegador también, es decir, todo esto se ha conseguido.

Se comentó que se añadiría una página web, esta también se terminó al completo.

Finalmente, se añadió un manual de instrucciones de forma adicional porque me parecía interesante y algo muy positivo para el juego y el proyecto en sí.

Resultados obtenidos:

En cuanto a los resultados obtenidos me parecen positivos ya que como comenté anteriormente el juego se ha completado tal y como se planeó en su momento, de todas formas, como todos los resultados, creo que son mejorables y debería seguir trabajando en ellos y hacer que este proyecto se convierta en una marca propia y no dejen de asociarlo con juegos ya existentes.

12. Bibliografía

<https://docs.unity3d.com/es/2020.2/Manual/UnityManual.html> Manual de Unity.

<https://answers.unity.com/questions/274809/how-to-make-enemy-chase-player-basic-ai.html?page=2&pageSize=5&sort=votes> Para hacer que te persigan enemigos.

<https://www.youtube.com/watch?v=IT7vDqm4xiY> Unity para retrasados 1.

<https://www.youtube.com/watch?v=DctCSiWHUIo> Unity para retrasados 2.

https://www.youtube.com/watch?v=uMRwq_AmIWg Unity para retrasados 3.

<https://www.youtube.com/watch?v=GbmRt0wydQU> Aprender a crear juegos desde 0.

<https://www.youtube.com/watch?v=JMT-tgtTKK8> Otro tutorial para crear juegos desde 0.

<https://www.youtube.com/watch?v=Ii-scMenaOQ&list=PLrnPJCHvNZuCVTz6lvhR81nnaf1a-b67U> Curso para hacer un videojuego en 2D.

https://www.youtube.com/watch?v=5o9b_VdjfDU Movimiento del player.

<https://www.youtube.com/watch?v=8C9h4CCoC2E&list=PLNEAWvYbJJ9m2sPYTv8pzkSklz79HoTRa> Para añadir Joystick.

<https://www.youtube.com/watch?v=zGvM2pM0QzA&list=PLNEAWvYbJJ9m2sPYTv8pkSklz79HoTRa&index=9> Poner un Start Menu.

<https://stackoverflow.com/questions/57010548/detect-and-follow-the-player/57010959>

Para que te persigan enemigos.

[Settings Menu in UNITY! 2021 Tutorial - YouTube](#) Hacer un Settings Menu en Unity.

[Como hacer ANIMACIONES en UNITY 2D - YouTube](#) Como hacer Animaciones de personajes en Unity.

[Flip enemy based on player direction Unity 2D - Unity Answers](#) Usado para Flippear enemigo.

[Tutorial - PISKEL - Sprites y animaciones en pixel art - YouTube](#) Tutorial que usé para aprender Piskel.

[How To Make 2D Teleporters In Unity - YouTube](#) Usado para crear el teletransportador.

[Countdown Timer In Unity - Easy Beginners Tutorial/ Guide - YouTube](#) Poner un contador de tiempo al videojuego.

[Unity 2018 - Platformer Tutorial 36: Powerups - YouTube](#) Para crear power ups.

[How to add a score counter into your Unity 2D game| Easy Unity 2D Tutorial - YouTube](#)

Para añadir un score counter al videojuego.

[Score Unity 2d \(for beginners\) - YouTube](#) añadir score counter. Otro tutorial.

13. Anexos

Enlaces de Interés (descargas del proyecto):

<https://github.com/pablohitos>

Códigos:

Los comentarios están disponibles en español también pero aparte. Una vez finalizado el proyecto me dediqué a traducirlos todos porque me comprometí que todo sería en inglés.

Player Move:

```
using System.Collections;
```

```

using System.Collections.Generic;
using UnityEngine;

public class PlayerMove : MonoBehaviour
{

    public float runSpeed = 1.75f; //variable for movement
on horizontal axis
    public float jumpSpeed = 3;
    public float doubleJumpSpeed = 2.5f; //double jump wont
be so big as the first one. In Y axis will have that speed
    private bool permitDoubleJump; /*this boolean variable
is the one that will let us know if we can make the double
jump or not so that when we are in the air we can make a
second jump*/

    Rigidbody2D rb2D; //reference to our Rigidbody for
everything related to physics, gravity, etc...

    /* some variables are added to have a similar jump in
mario bros game
     * if you push the jump button in a period short of time,
the jump wont be big
     * if you press long time the jump will be bigger*/

    public bool improvedJump = false; //to enable o o disable
the jump
    public float fallMultiplier = 0.5f;
    public float lowJumpMultiplier = 1f;
    public SpriteRenderer spriteRenderer; //al ser publica
podemos decirle cual queremos directamente desde el editor

    public Animator animator;

    public GameObject dustLeft; //left dust particle
    public GameObject dustRight; //right dust particle

    /*for dash we need a variable to know how much time has
passed since we can use a dash until the next one
     * a dash particle to know when we have been teleported
     * or made a dash to other side */

    public float dashCoolDown;
    public float dashForce = 30; //the speed of that dash
    public GameObject dashParticle;

    //for wall sliding we need boolean variables to know if
we touch the left wall o right wall
    //or if we are in wall sliding mode and to know the speed
that we are going down
}

```

```

bool isTouchingFront = false;
bool wallSliding;

public float wallSlidingSpeed = 0.75f;
bool isTouchingRight;
bool isTouchingLeft;

// Start is called before the first frame update
void Start()
{
    rb2D = GetComponent<Rigidbody2D>(); //reference to
player Rigidbody
}

private void Update() /*all checks related to doubleJump
and Jump are in the update because in
* fixeUpdate fails*/
{
    dashCoolDown -= Time.deltaTime;

    if (Input.GetKey("space") && wallSliding
==false) //we are gonna make it in this way for the doublejump
    { //an if that indicates that when we are on the
ground and we press space we will jump

        if (CheckGround.isGrounded)
        {
            permitDoubleJump = true; //at the moment of
jumping we activate doublejump
            rb2D.velocity = new
Vector2(rb2D.velocity.x, jumpSpeed); //before we were
changing X axis and then we change Y axis
        }
        else
        {
            if (Input.GetKeyDown("space"))/*we have to
add a getkeydown and when we press space bar we can make a
double jump*/
            {
                if (permitDoubleJump)
                {

                    animator.SetBool("DoubleJump",
true); //for activate the doublejump animation
                    rb2D.velocity = new
Vector2(rb2D.velocity.x, doubleJumpSpeed);
                    permitDoubleJump = false; // at the
moment of doublejump is in false
                }
            }
        }
    }
}

```

```

        }
    }

}

if (CheckGround.isGrounded == false)
{
    animator.SetBool("Jump", true); //with this we
know that we are not on the ground and we are jumping
    animator.SetBool("Run", false); //when player is
in the aire running is false

}

if (CheckGround.isGrounded == true)
{
    animator.SetBool("Jump", false); //when we are on
the ground jump is false
    animator.SetBool("DoubleJump", false); //at the
moment of being on the ground we want idle animation
    animator.SetBool("Falling", false); //at the
moment of being on the ground we want idle animation

}

if (rb2D.velocity.y < 0) //we know that player is
falling
{
    animator.SetBool("Falling", true);
}
else if (rb2D.velocity.y > 0) //el personaje esta
subiendo
{
    animator.SetBool("Falling", false);
}

/*we are going o check is we are touching a wall or
we are not on the ground we can do
WallSliding*/

if(isTouchingFront == true && CheckGround.isGrounded
== false)
{
    wallSliding = true;
}

else

```

```

    {
        wallSliding = false;
    }

    if (wallSliding)
    {
        animator.Play("Wall");
        //to mantain constant speed in sliding
        rb2D.velocity = new Vector2(rb2D.velocity.x,
Mathf.Clamp(rb2D.velocity.y,-wallSlidingSpeed,
float.MaxValue));
    }
}

private void OnTriggerEnter2D(Collider2D Powerup)
{
    if(Powerup.tag == "PowerupJump")
    {

        jumpSpeed = 5f;
        GetComponent<SpriteRenderer>().color =
Color.green;
        StartCoroutine(ResetPower());
    }

    if(Powerup.tag == "PowerupSpeed")
    {

        runSpeed = 3f;
        GetComponent<SpriteRenderer>().color =
Color.yellow;
        StartCoroutine(ResetPower());
    }
}

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

```

```

        //with this we are going to check which button are
        //pressing (ejemplo A, Flecha Izq,etc..)

        if (Input.GetKey("d") || Input.GetKey("right") &&
isTouchingRight==false)
        {
            rb2D.velocity = new Vector2(runSpeed,
rb2D.velocity.y); //with this we will make that when we press
the letter d our character moves, our rigidbody moves. With
the vector2 we are going to direction to the direction in
which we want to go
            spriteRenderer.flipX = false; //when we click on
the right the flip is deactivated
            animator.SetBool("Run", true); //is true
because at that moment we are moving the run animation is
activated.

            if (CheckGround.isGrounded == true)
                //this is to know if we are on the ground since
the particles will only activate when we are on the ground.
            {
                dustLeft.SetActive(true); //when we move to
the right, the dust particle on the left is activated.
                dustRight.SetActive(false); //the other
particle is deactivated

            }

            if (Input.GetKey("e") && dashCoolDown <= 0)
            {
                Dash(); //metodo dash
            }

        }

        else if (Input.GetKey("e") && dashCoolDown<=0)
        {
            Dash(); //metodo dash
        }

        else if (Input.GetKey("a") || Input.GetKey("left") &&
isTouchingLeft == false)
        {
            rb2D.velocity = new Vector2(-runSpeed,
rb2D.velocity.y); //this is the same as going to the right
but as we want to go to the left we set the runSpeed to
negative
        }
    }
}

```

```

        spriteRenderer.flipX = true; //when we press
left the flip is activated
        animator.SetBool("Run", true); //is true because
at that moment we are moving the run animation is activated.

        if (CheckGround.isGrounded == true)
            //this is to know if we are on the ground since
the particles will only activate when we are on the ground.

        {
            dustLeft.SetActive(false); //the other
particle is deactivated
            dustRight.SetActive(true); //when we move
to the left, the dust particle on the right is activated.

        }

        if (Input.GetKey("e") && dashCoolDown <= 0)
        {
            Dash(); //metodo dash
        }

    }

    else
    {
        rb2D.velocity = new Vector2(0, rb2D.velocity.y);
//in case any button is pressed and we want the player stay
still in his place
        animator.SetBool("Run", false); //no se activa
run al estar quieto

        dustLeft.SetActive(false); //as we are standing
still the dust particles are deactivated
        dustRight.SetActive(false); //as we are
standing still the dust particles are deactivated

    }

    if (improvedJump) //with this code we improve the
jump depeding the time we press. the player will jump more
or less
    {
        if (rb2D.velocity.y < 0)

```

```

        {
            rb2D.velocity
            += Vector2.up * Physics2D.gravity.y * (fallMultiplier) * Time.deltaTime;
        }
        if (rb2D.velocity.y > 0 &&
!Input.GetKey("space"))
        {
            rb2D.velocity += Vector2.up * Physics2D.gravity.y * (lowJumpMultiplier) * Time.deltaTime;
        }
    }

    public void Dash()
    {
        GameObject dashObject; /*we do this because then we
are going to use the destroy method to this specific object*/
        dashObject =
Instantiate(dashParticle, transform.position, transform.rotation);

        //to know if dash is made in left or right
        if(spriteRenderer.flipX == true)
        {
            rb2D.AddForce(Vector2.left * dashForce,
ForceMode2D.Impulse);
        }

        else
        {
            rb2D.AddForce(Vector2.right * dashForce,
ForceMode2D.Impulse);
        }

        dashCoolDown = 2; //we establish the cooldown again
Destroy(dashObject, 1);
    }

    //method to know if we are touching the right or left
    wall
    //it is to know continuously which wall we are touching

    private void OnCollisionStay2D(Collision2D collision)
{

```

```

        if
(collision.gameObject.CompareTag("RightSlidingWall"))
{
    isTouchingFront = true;
    isTouchingRight = true;
}

if
(collision.gameObject.CompareTag("LeftSlidingWall"))
{
    isTouchingFront = true;
    isTouchingLeft = true;
}

private void OnCollisionExit2D(Collision2D collision)
{
    isTouchingFront = false;
    isTouchingRight = false;
    isTouchingLeft = false;
}

private IEnumerator ResetPower()
{
    yield return new WaitForSeconds(10);
    jumpSpeed = 3;
    runSpeed = 1.75f;
    GetComponent<SpriteRenderer>().color = Color.white;
}

}

```

Player Move JoyStick:

```

using System.Collections;
using System.Collections.Generic;

```

```

using UnityEngine;

public class PlayerMoveJoystick : MonoBehaviour

{
    //we need a reference to the joystick and know when we
    move it left or right
    private float horizontalMove = 0f; //to know when move
    left or right. it will give a value which will be saved in
    this variable
    private float verticalMove = 0f; //the same but in
    vertical way

    public Joystick joystick; // reference to joystick

    public float runSpeed = 1.25f; //movement variable in
    an horizontal axis
    public float runSpeedHorizontal = 2; //movement
    variable in an horizontal axis

    public float jumpSpeed = 3;
    public float doubleJumpSpeed = 2.5f; //double jump won't
    be so big as the first one. In Y axis will have that speed
    .
    private bool permitDoubleJump; /*this boolean variable is
    the one that will let us know if we can make the double jump
    or not so that when we are in the air we can make a second
    jump*/
}

Rigidbody2D rb2D; //reference to our Rigidbody for
everything related to physics, gravity, etc...

/* some variables are added to have a similar jump in Mario
   Bros game
   * if you push in a short time the jump button the
   jump it won't be big
   * if you press long time the jump will be bigger*/

public SpriteRenderer spriteRenderer; //al ser publica
podemos decirle cual queremos directamente desde el editor

public Animator animator;

// Start is called before the first frame update
void Start()
{
    rb2D = GetComponent<Rigidbody2D>(); //reference to
    player Rigidbody
}

```

```

    }

    private void Update() /*all checks related to doubleJump
and Jump are in the update because in
* fixeUpdate fails*/
{

    if (horizontalMove > 0)
    {
        spriteRenderer.flipX = false; //when we
click on the right the flip is deactivated
        animator.SetBool("Run", true); //is true
because at that moment we are moving the run animation is
activated.

    }

    else if (horizontalMove < 0)
    {
        spriteRenderer.flipX = true; //when we click
on the left the flip is activated
        animator.SetBool("Run", true); //is true because
at that moment we are moving the run animation is activated.

    }

    else
    {
        // rb2D.velocity = new Vector2(0,
rb2D.velocity.y); //esto es en caso de que no pulsemos nada
y queremos que esté quieto en su sitio
        animator.SetBool("Run", false); //no run
activation when standing still
    }

}

if (CheckGround.isGrounded == false)
{
    animator.SetBool("Jump", true); //with this we
know that the moment we are not on the ground we are jumping.
    animator.SetBool("Run", false); //in the air
running will be false

}

if (CheckGround.isGrounded == true)

```

```

        {
            animator.SetBool("Jump", false); //at the moment
            of the ground, jump is false
            animator.SetBool("DoubleJump", false); //at the
            moment of being in the ground we want idle animation
            animator.SetBool("Falling", false); //at the
            moment of being in the ground we want idle animation
        }

        if (rb2D.velocity.y < 0) //we know that player is
        falling
        {
            animator.SetBool("Falling", true);
        }
        else if (rb2D.velocity.y > 0) //el personaje esta
        subiendo
        {
            animator.SetBool("Falling", false);
        }

    }

private void OnTriggerEnter2D(Collider2D Powerup)
{
    if (Powerup.tag == "PowerupJump")
    {

        jumpSpeed = 5f;
        GetComponent<SpriteRenderer>().color =
        Color.green;
        StartCoroutine(ResetPower());
    }

    if (Powerup.tag == "PowerupSpeed")
    {

        runSpeed = 3f;
        GetComponent<SpriteRenderer>().color =
        Color.yellow;
        StartCoroutine(ResetPower());
    }
}

// Update is called once per frame

```

```

    void FixedUpdate()
    {

        //El movimiento del Joystick
        horizontalMove      =      joystick.Horizontal      *
runSpeedHorizontal;
        //ahora le indicamos que mueva el peronaje
        transform.position += new Vector3(horizontalMove, 0,
0) * Time.deltaTime * runSpeed;

        //with this we are going to check which button are
pressing (ejemplo A, Flecha Izq,etc..)

    }

    public void Jump() //we put this public for making a
reference and then we wont have any problem in the jump in
android
        //this is the logic to put the jump
button
    {

        if (CheckGround.isGrounded)
        {
            permitDoubleJump = true; //at the moment of
jumping we activate doublejump
            rb2D.velocity = new Vector2(rb2D.velocity.x,
jumpSpeed); //before we were chaning x axis and then we
change y axis
        }
        else
        {

            if (permitDoubleJump)
            {

                animator.SetBool("DoubleJump", true);
//activate double jump animation
                rb2D.velocity = new
Vector2(rb2D.velocity.x, doubleJumpSpeed);
            }
        }
    }
}

```

```

        permitDoubleJump = false; // at the moment
of make double jump is in false
    }

}

private IEnumerator ResetPower()
{
    yield return new WaitForSeconds(10);
    jumpSpeed = 3;
    runSpeed = 2;
    GetComponent<SpriteRenderer>().color = Color.white;

}

```

Player Respawn:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

```

```

using UnityEngine.Audio;

public class PlayerRespawn : MonoBehaviour
{
    public AudioSource clip; //esta es la referencia para el audio

    //referencia de las vidas
    public GameObject[] hearts; //array donde van las vidas
    private int life;

    private float checkPointPositionX, checkPointPositionY;
    public Animator animator;

    private void Start()
    {
        life = hearts.Length;

        if (PlayerPrefs.GetFloat("checkPointPositionX") != 0) /* en el momento que sea distinto de 0 es que hemos asignado * algo en el argumento de ReachedPoint*/
        {
            //Hemos sabido que hemos pasado al player por el Checkpoint por lo que podremos mandarlo a esa posición
            transform.position = (new Vector2(PlayerPrefs.GetFloat("checkPointPositionX"),
            PlayerPrefs.GetFloat("checkPointPositionY")));
        }
    }

    private void Checklife()
    {
        if (life<1)
        {
            Destroy(hearts[0].gameObject);

            SceneManager.LoadScene(SceneManager.GetActiveScene().name);
            //reseteamos el nivel una vez dañado
        }
        else if (life < 2)
        {
    
```

```

        Destroy(hearts[1].gameObject);
        animator.Play("Hit");//se activa en el momento
en el que player es dañado
        clip.Play();

    }
    else if(life < 3)
    {
        Destroy(hearts[2].gameObject);
        animator.Play("Hit");//se activa en el momento
en el que player es dañado
        clip.Play();

    }
    else if(life < 4)
    {
        Destroy(hearts[3].gameObject);
        animator.Play("Hit");//se activa en el momento
en el que player es dañado
        clip.Play();

    }
}

```

```

public void ReachedCheckPoint(float x, float y)//cuando
hayamos pasado por el checkpoint, estamos pasando la posicion
del checkpoint
{
    /* va a guardar la
    posicion del checkpoint
    * para cuando le demos
    al play aparecer ahí*/
    PlayerPrefs.SetFloat("checkPointPositionX", x);
    //con esto guardamos info, guardamos la posicion en el eje
    x.
    PlayerPrefs.SetFloat("checkPointPositionY",
    y); //con esto guardamos info, guardamos la posicion en el
    eje y.

}

public void PlayerDamaged() //lo llamaremos desde los
enemigos
{
    life--;
    Checklife();
}

```

Player Select:

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

```

public class PlayerSelect : MonoBehaviour
{
    public bool enableSelectCharacter;
    public enum Player {MaskedMan, PinkMan, NinjaFrog,
VirtualGuy, Squirrell, Rabbit, Fox } //for select the
character
    public Player playerSelected; //this will permit select
the player

    public Animator animator;
    public SpriteRenderer spriteRenderer;

    public RuntimeAnimatorController[] playersController;
/*to get the tree animations we will use an array because we
will have more than

* *an object of that type*/
    public Sprite[] playersRenderer; //reference to sprites

    // Start is called before the first frame update
    void Start()
    {

        if (!enableSelectCharacter) // si no activamos esta
forma activaremos la forma de ChangeSkin para cambiar de
personaje
        {
            //Para no cambiar el personaje desde el inspecto
de unity
            //creamos un metodo para eso
            ChangePlayerInMenu();
        }

        else
        {
            switch (playerSelected)
            {
                case Player.MaskedMan:
                    spriteRenderer.sprite = 
playersRenderer[0];
                    animator.runtimeAnimatorController = 
playersController[0];

                    break;
                case Player.PinkMan:
                    spriteRenderer.sprite = 
playersRenderer[1];
                    animator.runtimeAnimatorController = 
playersController[1];
            }
        }
    }
}

```

```

        break;
    case Player.NinjaFrog:
        spriteRenderer.sprite      =
playersRenderer[2];
        animator.runtimeAnimatorController      =
playersController[2];

        break;
    case Player.VirtualGuy:

        spriteRenderer.sprite      =
playersRenderer[3];
        animator.runtimeAnimatorController      =
playersController[3];
        break;
    case Player.Squirrel:
        spriteRenderer.sprite      =
playersRenderer[4];
        animator.runtimeAnimatorController      =
playersController[4];

        break;
    case Player.Rabbit:
        spriteRenderer.sprite      =
playersRenderer[5];
        animator.runtimeAnimatorController      =
playersController[5];

        break;
    case Player.Fox:
        spriteRenderer.sprite      =
playersRenderer[6];
        animator.runtimeAnimatorController      =
playersController[6];

        break;
    default:
        break;
    }
}

}

public void ChangePlayerInMenu() //method for change the
player in game
{
    switch (PlayerPrefs.GetString("PlayerSelected"))

```

```

    {
        case "MaskedMan":
            spriteRenderer.sprite = playersRenderer[0];
            animator.runtimeAnimatorController =
            playersController[0];

            break;
        case "PinkMan":
            spriteRenderer.sprite = playersRenderer[1];
            animator.runtimeAnimatorController =
            playersController[1];

            break;
        case "NinjaFrog":
            spriteRenderer.sprite = playersRenderer[2];
            animator.runtimeAnimatorController =
            playersController[2];

            break;
        case "VirtualGuy":

            spriteRenderer.sprite = playersRenderer[3];
            animator.runtimeAnimatorController =
            playersController[3];
            break;
        case "Squirrel":
            spriteRenderer.sprite = playersRenderer[4];
            animator.runtimeAnimatorController =
            playersController[4];

            break;
        case "Rabbit":
            spriteRenderer.sprite = playersRenderer[5];
            animator.runtimeAnimatorController =
            playersController[5];

            break;
        case "Fox":
            spriteRenderer.sprite = playersRenderer[6];
            animator.runtimeAnimatorController =
            playersController[6];

            break;
        default:
            break;
    }

}

```

```
}
```

UI Manager:

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

```

using TMPro;
using UnityEngine.SceneManagement;
using UnityEngine.Audio; //volume control

public class UIManager : MonoBehaviour
{
    public GameObject optionsPanel;
    public GameObject otherOptions;
    public AudioMixer mainMixer;
    public AudioSource clip;

    public void OptionsPanel()
    {
        Time.timeScale = 0; //to stop time in game
        optionsPanel.SetActive(true); //for panel activation
    }

    public void Return()
    {
        Time.timeScale = 1; //for time activation
        optionsPanel.SetActive(false); //disabled main panel
        otherOptions.SetActive(false); //enable panel
    }

    public void OtherOptions()
    {
        Time.timeScale = 0; //to stop time in game
        otherOptions.SetActive(true); //enable panel
        optionsPanel.SetActive(false); //disable previous panel
        //sound
        //music
        //Graphics
    }

    public void SetFullscreen(bool isFullscreen) //for full screen
    {
        Screen.fullScreen = isFullscreen;
    }

    public void SetQuality(int qualityIndex)
    {
        QualitySettings.SetQualityLevel(qualityIndex);
    }
}

```

```

public void SetVolume(float volume)
{
    mainMixer.SetFloat("volume", volume);
}

public void GoMainMenu() //back to main menu
{
    SceneManager.LoadScene("SelectLevel");
    Time.timeScale = 1; //enable time
}

public void QuitGame() //exit game
{
    Application.Quit();
    Time.timeScale = 1; //activate time
}

//method for sound when we press buttons
public void PlaySoundButton()
{
    clip.Play();
}

```

AI Chaising:

```

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

```

```

public class AiChasing : MonoBehaviour
{
    public float speed;
    public GameObject player;
    private Vector2 actualPos; //current position
    public SpriteRenderer spriteRenderer; /*reference to the
    spriterenderer to change the flipX */
    public Animator animator; /*reference to our
    animator to change the animation of the enemy, whether he is
    walking or being hit etc*/

    void Update()
    {

        Vector3 localPosition = player.transform.position -
        transform.position;
        localPosition = localPosition.normalized; // The
        normalized direction in LOCAL space
        transform.Translate(localPosition.x *
        Time.deltaTime * speed, localPosition.y * Time.deltaTime * *
        speed, localPosition.z * Time.deltaTime * speed);

        this.spriteRenderer.flipX =
        player.transform.position.x > this.transform.position.x;
        //this is the flip if it is on its back it will have one
        orientation and if it is in front it will have another
        orientation
    }

}

```

Angry Pig Ai and Jump:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Audio;

```

```

public class AngryPigAiAndJump : MonoBehaviour
{
    public AudioSource clip;//this is the reference for the
audio

    //AI part

    public Animator animator; /*reference to our animator to
change the animation of the enemy, whether he is walking or
being hit etc*/
    public SpriteRenderer spriteRenderer; /*reference to the
spriterenderer to change the flipX so that if it
it goes to one
side make flip in the animation or not*/

    public float speed = 0.5f; //the enemy's displacement
speed
    private float waitTime;//the time the enemy will stay
when it reaches a certain point
    public float startWaitTime = 2;//the time will reamin in
certain point
    private int i = 0; //to manage the point where he is and
the where he has to go
    private Vector2 actualPos; //actual position
    public Transform[] moveSpots; //to have the reference of
the point where the enemy has to go

    //Jump Damage Part
    public Collider2D collider2D;
    public SpriteRenderer spriterenderer;
    public GameObject destroyParticle;
    public float jumpForce = 2.5f;
    public int lifes = 3;

//Start of AI part

    // Start is called before the first frame update
    void Start()
    {
        waitTime = startWaitTime; //it is a way to initialize
them
    }

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

```

```

    {
        StartCoroutine(CheckEnemyMoving());

        //here where going to work with the movement of the
        enemy when he has to go towards certain point
        //he will move towards certain direction, with
        velocity, position, now we proceed to specify
        transform.position = Vector2.MoveTowards(transform.position,
        moveSpots[i].transform.position, speed * Time.deltaTime);

        //to change to the next point
        if (Vector2.Distance(transform.position,
        moveSpots[i].transform.position) < 0.1f)
        {
            //timer to keep or mantain in idle
            if (waitTime <= 0)
            {
                // if the time has elapsed or completed, you
                can move on to the next point.
                //to check if there are more points to go
                to.

                if (moveSpots[i] != moveSpots[moveSpots.Length - 1])
                {
                    // if we have more points to go to, we
                    increase the waypoint.
                    i++;
                }
                // if there are no more points we start again
                from point 1.
                else
                {
                    i = 0;
                }
                // restart
                waitTime = startWaitTime;
            }
        }
        else
        {
            //decreasing time
            waitTime -= Time.deltaTime;
        }
    }

    IEnumerator CheckEnemyMoving() //it is a corrutine

```

```

    {
        //to know in which position we are
        actualPos = transform.position;
        //we wait and specify the time until we pass to the
        next line of code
        yield return new WaitForSeconds(0.5f);
        //this is to change the flip
        if (transform.position.x > actualPos.x)
        //to know if we move to the right
        {
            spriteRenderer.flipX = true;
            animator.SetBool("Idle", false); //it is in
false because the enemy is standstill
        }

        else if (transform.position.x < actualPos.x)
        //to know if we move to the left
        {
            spriteRenderer.flipX = false;
            animator.SetBool("Idle", false); //it is in
false because the enemy is standstill
        }
        else if (transform.position.x == actualPos.x)
        {
            //after 0.5 seconds we set or establish idle to
true
            animator.SetBool("Idle", true);
        }
    }

    //Start of Jump Part
    private void OnCollisionEnter2D(Collision2D collision) //to know if we collide with the player
    {

        if (collision.transform.CompareTag("Player")) //to
know if we collide with the player
        {
            //with this we make when we collide with the
player he will jump a little bit

            collision.gameObject.GetComponent<Rigidbody2D>().velocity =
(Vector2.up * jumpForce);
            LoseLifeAndHit(); //method to know if we lost
life
            CheckLife(); //method to know if the enemy has
died
        }
    }

```

```

}

public void LoseLifeAndHit()
{
    lifes--;//for subtract one life
    animator.Play("Hit"); //the animation of hit
    //we lost life and we active hit
}

public void CheckLife()

{
    if (lifes <= 2 || lifes <= 1 || lifes <= 0)
    {
        speed = 1.00f; //we increase the life when the
        enemy only has 2 lifes
        animator.Play("Run"); //activate the running
        animation
    }

    if (lifes == 0)
    {

        destroyParticle.SetActive(true); //if our lifes
        is equal to 0
        spriterenderer.enabled = false;
        Invoke("EnemyDie", 0.2f); //invocar method and
        0.2 seg time
        clip.Play();
    }
}

public void EnemyDie() //method to dead enemy
{
    Destroy(gameObject);
}

}

```

Back Button:

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

```

using UnityEngine.Audio; //control volume

public class BackButton : MonoBehaviour
{
    public AudioSource clip;

    public void BackToMain()
    {
        SceneManager.LoadScene("MainScreen");
        clip.Play();
    }

    public void PlaySoundButton()
    {
        clip.Play();
    }
}

```

Checkground:

```

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

```

```

public class CheckGround : MonoBehaviour
{
    //we add a boolean variable
    public static bool isGrounded;
    //with this we will know if our player is on the floor
    or not

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Ground"))
        {
            isGrounded = true;
            /* we will know if we are on the ground when we
            will collide on it
            * OnTriggerEnter2D y we can check if is the tile map
            and not fruits*/
        }
    }

    } //we put static because we can use this variable inside
    in other script

    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.CompareTag("Ground"))
        {
            isGrounded = false;
        }
    }
}

```

Checkpoint:

```
using System.Collections;
```

```

using System.Collections.Generic;
using UnityEngine;

public class CheckPoint : MonoBehaviour
{

    private void OnTriggerEnter2D(Collider2D collision)
//with this script we are going to check how many times the
player has collided

    {
        if (collision.CompareTag("Player"))/*when the check
point collides with the tag player we save the position, so
when we press play the player will be on that position */
        {

            collision.GetComponent<PlayerRespawn>().ReachedCheckPoint(t
ransform.position.x,      transform.position.y);      //estamos
pasando la posicion en x e y del checkpoint

                GetComponent<Animator>().enabled = true; /* when
player goes by the checkpoint will be activated */

        }

    }

}

```

Damage Object:

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

public class DamageObject : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {

        if (collision.transform.CompareTag("Player")) //with
this we will know if we collide with the player
        {
            Debug.Log("Player Damaged");

collision.transform.GetComponent<PlayerRespawn>().PlayerDam
aged();
        }

    }
}
```

Enemy Spike

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

public class EnemySpike : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {

        if (collision.transform.CompareTag("Player")) //with
this we will know if the enemy collides with player
        {
            Debug.Log("Player Damaged");

collision.transform.GetComponent<PlayerRespawn>().PlayerDam
aged();
        }

    }
}
```

Fruit Manager:

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

```

using UnityEngine.UI;
using TMPro;
public class FruitManager : MonoBehaviour
{

    public TextMeshProUGUI LevelCleared;
    public TextMeshProUGUI totalFruits; //text variable for
total fruits of the level
    public TextMeshProUGUI pickedFruits; //text variable for
picked fruits

    public GameObject transition; // to active the transition
in the correct moment

    public AudioSource clip;

    private int totalFruitsInlevel;

    int LevelLock;
    public Button[] levelButtons;

    private void Start()
    {
        /*we are going to specify that totalfruits is equal to
childcount*/
        totalFruitsInlevel = transform.childCount;

        LevelLock = PlayerPrefs.GetInt("LevelLock", 1);

        for (int i = 0; i < levelButtons.Length; i++)
        {

            levelButtons[i].interactable = false;
        }

        for(int i = 0; i < LevelLock; i++)
        {
            levelButtons[i].interactable = true;
        }
    }

    public void openLevel(int LevelIndex)
    {
        SceneManager.LoadScene(LevelIndex);
    }
}

```

```

    //with this script we are going to ask we have any fruits
left , if we have any child left
    private void Update()
    {
        AllFruitsCollected(); //this is to ask how many
fruits it has
        totalFruits.text = totalFruitsInlevel.ToString();
/*to show how many fruits we have taken or how many

* there are in total. These are the fruits of the whole
level. We modify

* the integer, we show it in the screen using a in String*/
        pickedFruits.text =
transform.childCount.ToString(); //total fruits left in
fruit manager

    }

    public void AllFruitsCollected() // this method doesnt
give back anything

    {

        if(transform.childCount == 0)
        {

            Debug.Log("There are no more Fruits,
Congratulations!");
            LevelCleared.gameObject.SetActive(true);
//activate text
            transition.SetActive(true);

            Invoke("ChangeScene", 4); /*we invoke the method
to change the scene. With this the scene wont
                                         * change very fast so
we can see the text of leve completed
                                         */
        }

    }

```

```

    void ChangeScene()
    {
        int currentLevel = SceneManager.GetActiveScene().buildIndex;
        PlayerPrefs.SetInt("LevelLock", currentLevel);

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}

```

Fruits Collected

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Audio;//this library is for the audio

```

```

public class FruitsCollected : MonoBehaviour
{
    public AudioSource clip; //this is the reference for the audio

    //we need to create we something contacts with the collider

    private void OnTriggerEnter2D(Collider2D collision)
    {
        //we have to know when collides with the player

        if (collision.CompareTag("Player"))
            /*player tag will help with the picking up fruits issue.
             * when tag players collides with the fruit, the fruit animation will
             * disspear or will be disabled, then the fruit will show up as picked up
             * in the fruit counter
             */
        {
            GetComponent<SpriteRenderer>().enabled = false;
            //to disable the fruit (apple, banana, etc..) sprite

            gameObject.transform.GetChild(0).gameObject.SetActive(true);
            //

            //the gameobject colleted will be activated when the fruit is picked up
            //we take the child and we activate it
            //at the momeent of colliding with the fruit sprite, it will be disabled and the
            //componenet collected will be activated

            Destroy(gameObject, 0.5f); //with this this we disable the gameObject when the fruit
            //is picked up

            //when player collides with the fruit there is a sound
            clip.Play();
        }
    }
}

```

Instant Death:

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

```

public class InstantDeath : MonoBehaviour
{
    public void OnCollisionEnter2D(Collision2D collision)
    {
        if
(collision.transform.CompareTag("Player")) //this is the way
to know if the player collides with this
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().name);
//we reset the game when collides
        }
    }
}

```

Main Menu Screen Script:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

```

```

using UnityEngine.Audio; //volume control

public class MainMenuScreenScript : MonoBehaviour
{
    public AudioSource clip;

    public void MainMenuLevel()
    {
        SceneManager.LoadScene("SelectLevel");
        clip.Play();
    }

    public void PlaySoundButton()
    {
        clip.Play();
    }
}

```

Moving Platform

```

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

```

```

public class MovingPlatform : MonoBehaviour
{
    public float speed = 0.5f; //this is the enemy speed
movement
    private float waitTime; //this is the time the platform
will remain or will stay when it reaches to certain point

    public float startWaitTime = 2; //the time will stay or
remain in certain point
    private int i = 0; //to manage the point where he is and
the where he has to go
    public Transform[] moveSpots; //to have the reference of
the point where the platform has to go

    // Start is called before the first frame update
    void Start()
    {
        waitTime = startWaitTime; //it is a way to inizialice
    }

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

        //we are going to work with the movement of the
platform towards a point
        //it will move to a direction with a position, speed
and specify point
        transform.position =
Vector2.MoveTowards(transform.position,
moveSpots[i].transform.position, speed * Time.deltaTime);

        //change to the next point
        if (Vector2.Distance(transform.position,
moveSpots[i].transform.position) < 0.1f)
        {
            //timer to stay in idle
            if (waitTime <= 0)
            {
                //if time is completed the platform can pass
                to the next point
                //it check if there are more points to go

                if (moveSpots[i] !=
moveSpots[moveSpots.Length - 1])
                {
                    //if we have more points to go we
increase the waypoints
                    i++;
                }
            }
        }
    }
}

```

```

                //if there no more points to go we back to
start from the point 1
            else
            {
                i = 0;
            }
            //we restart it
            waitTime = startWaitTime;
        }

        else
        {
            //we decrease the time
            waitTime -= Time.deltaTime;
        }

    }

private void OnCollisionEnter2D(Collision2D collision)
{
    collision.collider.transform.SetParent(transform);
}

private void OnCollisionExit2D(Collision2D collision)
{
    collision.collider.transform.SetParent(null);
}

//With this code the character will be able to join the
platforms without any problem,
//it will become a child of the platform and will be
perfectly attached without physics problems.

}

```

AI Basic:

```

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

public class AIBasic : MonoBehaviour

```

```

{
    public Animator animator; /*reference to our animator to
change the animation of the enemy, whether he is walking or
being hit etc*/
    public SpriteRenderer spriteRenderer; /*reference to the
spriterenderer to change the flipX so that if it
it goes to one
side make flip in the animation or not*/

    public float speed = 0.5f; //the enemy's displacement
speed
    private float waitTime;//the time the enemy will stay
when it reaches a certain point

    public float startWaitTime = 2;//the time will reamin in
certain point
    private int i = 0; //to manage the point where he is and
the where he has to go
    private Vector2 actualPos; //actual position
    public Transform[] moveSpots; //to have the reference of
the point where the enemy has to go

    // Start is called before the first frame update
    void Start()
    {
        waitTime = startWaitTime; //it is a way to initialize
them
    }

    // Update is called once per frame
    void Update()
    {
        StartCoroutine(CheckEnemyMoving());
        //here where going to work with the movement of the
enemy when he has to go towards certain point
        //he will move towards certain direction, with
velocity, position, now we proceed to specify
        transform.position =
Vector2.MoveTowards(transform.position,
moveSpots[i].transform.position, speed * Time.deltaTime);

        //to change to the next point
        if (Vector2.Distance(transform.position,
moveSpots[i].transform.position) < 0.1f)
        {
            //timer to keep or mantain in idle
            if (waitTime <= 0)
            {

```

```

                // if the time has elapsed or completed, you
can move on to the next point.
                //to check if there are more points to go
to.

                if (moveSpots[i] != moveSpots[moveSpots.Length - 1])
                {
                    // if we have more points to go to, we
increase the waypoint.
                    i++;
                }
                // if there are no more points we start again
from point 1.
                else
                {
                    i = 0;
                }
                // restart
                waitTime = startWaitTime;
            }

            else
            {
                //decreasing time
                waitTime -= Time.deltaTime;
            }
        }

    }

IEnumerator CheckEnemyMoving() //it is a coroutine
{
    //to know in which position we are
    actualPos = transform.position;
    //we wait and specify the time until we pass to the
next line of code
    yield return new WaitForSeconds(0.5f);
    //this is to change the flip
    if (transform.position.x > actualPos.x)
    //to know if we move to the right
    {
        spriteRenderer.flipX = true;
        animator.SetBool("Idle", false); //it is in
false because the enemy is standstill
    }

    else if (transform.position.x < actualPos.x)
    //to know if we move to the left
    {

```

```

        spriteRenderer.flipX = false;
        animator.SetBool("Idle", false); //it is in
false because the enemy is standstill
    }
    else if(transform.position.x==actualPos.x)
    {
        //after 0.5 seconds we set or establish idle to
true
        animator.SetBool("Idle", true);
    }
}

```

Jump Damage

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Audio;

```

```

public class JumpDamage : MonoBehaviour
{
    public AudioSource clip; //esta es la referencia para el audio

    public Animator animator;
    public SpriteRenderer spriterenderer;
    public GameObject destroyParticle;
    public float jumpForce = 2.5f;
    public int lifes = 2;

    private void OnCollisionEnter2D(Collision2D collision) //to know if we collide with the player
    {
        if (collision.transform.CompareTag("Player")) //to know if we collide with the player
        {
            //with this we make when we collide with the player he will jump a little bit

            collision.gameObject.GetComponent<Rigidbody2D>().velocity =
            (Vector2.up * jumpForce);
            LoseLifeAndHit(); //method to know if we lost life
            CheckLife(); //method to know if the enemy has died
        }
    }

    public void LoseLifeAndHit()
    {
        lifes--; // subtract one life
        animator.Play("Hit"); //the animation of hit
        //we lost life and we active hit
    }

    public void CheckLife()
    {
        if (lifes == 0)
        {
            destroyParticle.SetActive(true); //if our lifes is equal to 0
        }
    }
}

```

```

        spriterenderer.enabled = false;
        clip.Play();
        Invoke("EnemyDie", 0.2f); //invocar method and
0.2 seg time

    }

}

public void EnemyDie() //method to dead enemy
{

    Destroy(gameObject);

}

```

Plant Enemy:

```

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

```

```

public class PlantEnemy : MonoBehaviour
{
    private float waitedTime; //to know how many time we
    have waited since we fired the first bullet until the second
    bullet
    public float waitTimeToAttack = 3; //each 3 second a
    bullet is fired
    public Animator animator; // reference to animator to
    enable the attack animation
    public GameObject bulletPrefab; // reference to
    bulletprefab
    public Transform launchSpawnPoint; // position where the
    bullet is fired
                                                //the enemy is
    shooting all the time each 2 o 3 seconds

    private void Start()
    {
        waitedTime = waitTimeToAttack; //the time that will
        pass
    }

    private void Update()
    {
        if(waitedTime <= 0) // if the time is less than 0 we
        will attack
        {
            waitedTime = waitTimeToAttack; //we equalize so
            it wont attack all the time
            animator.Play("Attack"); //activation attack
            animation
            Invoke("LaunchBullet", 0.5f); //invoke method
        }

        else
        {
            waitedTime -= Time.deltaTime; //this substract
            3 seconds to waitedTime
            //this is create a cicle. Each 3 seconds attacks
        }
    }

    public void LaunchBullet()
    {

        //reference to that bullet
        GameObject newBullet;
    }
}

```

```
        newBullet      = Instantiate(bulletPrefab,  
launchSpawnPoint.position, launchSpawnPoint.rotation);  
        //instance to the bullet in a specified position  
    }  
}
```

Bullet Plant:

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

```

using UnityEngine;

public class BulletPlant : MonoBehaviour
{
    // for the bullet we need to know the direction
    // how much time pass for his destruction

    public float speed = 2; //for his speed
    public float lifeTime = 2; //destroy bullet when some
time is passed
    public bool left; // para activar la orientacion de la
planta a la izquierda o derecha

    private void Start()

    {
        Destroy(gameObject, lifeTime); // ponemos el destroy
en el Start para destruir la bala
    }

    private void Update()
    {
        if (left)
        {
            transform.Translate(Vector2.left * speed *
Time.deltaTime);
        }
        else
        {
            transform.Translate(Vector2.right * speed *
Time.deltaTime);
        }
    }
}

```

Zombee:

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

public class Zombee : MonoBehaviour
{
    public Animator animator; // reference to animator to
enable the attack animation
    public SpriteRenderer spriteRenderer; /*reference to the
spriterenderer to change the flipX so that if it
it goes to one
side make flip in the animation or not*/

    public float speed = 0.5f; //the enemy's displacement
speed
    private float waitTime;//the time the enemy will stay
when it reaches a certain point

    public float startWaitTime = 2;//the time will reamin in
certain point
    private int i = 0; //to manage the point where he is and
the where he has to go
    private Vector2 actualPos; //actual position
    public Transform[] moveSpots; //to have the reference of
the point where the enemy has to go

    // Start is called before the first frame update
    void Start()
    {
        waitTime = startWaitTime; //it is a way to initialize
them
    }

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

        //here where going to work with the movement of the
enemy when he has to go towards certain point
        //he will move towards certain direction, with
velocity, position, now we proceed to specify
        transform.position =
Vector2.MoveTowards(transform.position,
moveSpots[i].transform.position, speed * Time.deltaTime);

        //to change to the next point
        if (Vector2.Distance(transform.position,
moveSpots[i].transform.position) < 0.1f)
    {
```

```

        //timer to keep or mantain in idle
        if (waitTime <= 0)
        {
            // if the time has elapsed or completed, you
            can move on to the next point.
            //to check if there are more points to go
            to.

            if (moveSpots[i] != moveSpots[moveSpots.Length - 1])
            {
                // if we have more points to go to, we
                increase the waypoint.
                i++;
            }
            // if there are no more points we start again
            from point 1.
            else
            {
                i = 0;
            }
            // restart
            waitTime = startWaitTime;
        }

        else
        {
            //decreasing time
            waitTime -= Time.deltaTime;
        }

    }

}

```

Zombee Attack

```

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

```

```

public class ZombeeAttack : MonoBehaviour
{
    public Animator animator;
    public float distanceRaycast = 0.5f;
    private float cooldownAttack = 1.5f; //a kind of delay
    so that it doesn't attack us all the time
    private float actualCoolDownAttack; // puts it 0 from
    the beginning
    public GameObject beeBullet;

    // Start is called before the first frame update
    void Start()
    {
        actualCoolDownAttack = 0;
    }

    // Update is called once per frame
    void Update()
    {
        actualCoolDownAttack -= Time.deltaTime; //when the
        currentCooldown is 0 you can attack
    }

    private void FixedUpdate()
    {
        //here we start with RayCast
        RaycastHit2D hit2D = Physics2D.Raycast(transform.position, Vector2.down,
        distanceRaycast);
        //here we are setting up the beam, we tell it where
        it has to go

        //we ask why the beam has collided

        if (hit2D.collider != null)
        {
            // we check that the lightning collided with the
            player
            if (hit2D.collider.CompareTag("Player"))
            {
                if (actualCoolDownAttack < 0) //when it is
                less than 0, the bullet is launched.
                {
                    Invoke("LaunchBullet", 0.5f);
                    animator.Play("Attack");
                    actualCoolDownAttack = cooldownAttack;
                }
            }
        }
    }
}

```

```

        }

    }

    void LaunchBullet()
    {
        GameObject newBullet;

        newBullet = Instantiate(beeBullet,
transform.position, transform.rotation);
    }

}

```

Change Skin:

```

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

```

```

using UnityEngine.Audio; //para el control del volumen

public class ChangeSkin : MonoBehaviour

{
    public GameObject skinChange;
    public GameObject player;
    public AudioMixer mainMixer;
    public AudioSource clip;

    public void SkinChange()
    {
        skinChange.SetActive(true); //para activar el panel
        clip.Play();
    }

    /**necesitamos unos 7 metodos para cambiar los skins lo
    ponemos de tipo publico
    una vez seleccionado el skin se guarda el skin en el
    PlayerPrefs, asi se mantiene en el main menu
    , cerraremos el panel y se pondrá el skin en específico.
    Por lo que creamos un metodo que se encargue
    de todo eso*/
    public void SetPlayerMaskedMan()
    {
        PlayerPrefs.SetString("PlayerSelected",
        "MaskedMan");
        ResetPlayerSkin();
        clip.Play();
    }

    public void SetPlayerPinkMan()
    {
        PlayerPrefs.SetString("PlayerSelected", "PinkMan");
        ResetPlayerSkin();
        clip.Play();
    }

    public void SetPlayerNinjaFrog()
    {
        PlayerPrefs.SetString("PlayerSelected",
        "NinjaFrog");
        ResetPlayerSkin();
        clip.Play();
    }
}

```

```

}

public void SetPlayerVirtualGuy()
{
    PlayerPrefs.SetString("PlayerSelected",
"VirtualGuy");
    ResetPlayerSkin();
    clip.Play();
}

public void SetPlayerSquirrell()
{
    PlayerPrefs.SetString("PlayerSelected",
"Squirrell");
    ResetPlayerSkin();
    clip.Play();
}

public void SetPlayerRabbit()
{
    PlayerPrefs.SetString("PlayerSelected", "Rabbit");
    ResetPlayerSkin();
    clip.Play();
}

public void SetPlayerFox()
{
    PlayerPrefs.SetString("PlayerSelected", "Fox");
    ResetPlayerSkin();
    clip.Play();
}

//hacemos el metodo de cambiar skin

void ResetPlayer()
{
skinChange.gameObject.SetActive(false); //desactivamos el
skin panel
}

```

```

//necesitamos un metodo para poner el skin a tiempo real

player.GetComponent<PlayerSelect>().ChangePlayerInMenu();
}

void ResetPlayerSkin()
{
    skinChange.SetActive(false);      //desactivamos el
skinpanel
}

public void PlaySoundButton()
{
    clip.Play();
}

// Start is called before the first frame update
void Start()
{

}

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

```

JumpBox:

```

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

```

```

using UnityEngine.Audio;

public class JumpBox : MonoBehaviour
{
    public AudioSource clip; //esta es la referencia para el audio

    public Animator animator; //para llamar a la animacion de hit
    public SpriteRenderer spriteRenderer;
    public GameObject brokenParts; //donde estan las partes rotas
    public float jumpForce = 4f; //la fuerza con la que nos impulsa la caja cuando la estamos rompiendo

    public int lifes = 1; //la vida de la caja
    public GameObject boxCollider;
    public Collider2D col2D;

    public GameObject fruit;

    private void Start()
    {
        fruit.SetActive(false); //desactivar el objeto cuando activemos la funcion de hit
        //para llevar la fruta al fruit manager

        fruit.transform.SetParent(FindObjectOfType<FruitManager>().transform);
    }
}

private void OnCollisionEnter2D(Collision2D collision)
{
    //para saber si hemos colisionado con el player

    if (collision.transform.CompareTag("Player"))
        //colisionamos con el player y preguntamos si es el player ese este codigo
    {

        collision.gameObject.GetComponent<Rigidbody2D>().velocity =
        Vector2.up * jumpForce; //pega un pequeño salto hacia arriba cuando destruimos la caja
        LosesLifeAndHit(); //este metodo lo llamaremos una vez colisionado con la caja
    }
}

```

```

        }

    }

public void LosesLifeAndHit()
{
    lifes--;
    animator.Play("Hit");
    CheckLife(); //para comprobar si la vida es 0
    clip.Play();

}

public void CheckLife()
{
    if(lifes <= 0)
    {
        fruit.SetActive(true); //se activa la fruta
cuando destruya la caja y sus vidas sean 0
        boxCollider.SetActive(false);
        col2D.enabled = false;

        brokenParts.SetActive(true);
        spriteRenderer.enabled = false;
        Invoke("DestroyBox", 0.5f);
    }
}

public void DestroyBox()
{
    Destroy(transform.parent.gameObject);
}

}

```

Trampoline:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Audio;

```

```

public class Trampoline : MonoBehaviour
{
    public AudioSource clip; //esta es la referencia para el audio

    public Animator animator; //para poder activar la animacion
    public float jumpForce = 8f; // la fuerza con la que el player va a hacia arriba
    private void OnCollisionEnter2D(Collision2D collision)
    //sistema de colision
    {
        if (collision.transform.CompareTag("Player"))
        //aqui preguntamos si el player ha chocado el objeto
        {
            //si choca se activa una fuerza de empuje
            collision.gameObject.GetComponent<Rigidbody2D>().velocity = (Vector2.up * jumpForce);
            clip.Play();
            animator.Play("TrampolineJump"); //activamos la animacion del trampolin
        }
    }
}

```

Score Script:

```

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

```

```

public class ScoreScript : MonoBehaviour
{
    public TextMeshProUGUI MyScoreText;
    private int ScoreNumber;
    // Start is called before the first frame update
    void Start()
    {
        ScoreNumber = 0; //we initialize the score to zero
        MyScoreText.text = "Score:" + " " + ScoreNumber;

    }

    private void OnTriggerEnter2D(Collider2D CoinAndEnemy)
    //when player collision with coins
    {
        if(CoinAndEnemy.tag == "Coin1")
        {
            ScoreNumber += 10;

            MyScoreText.text = "Score:" + " " + ScoreNumber;
            //The scorenumber changes
        }

        if (CoinAndEnemy.tag == "Coin2")
        {
            ScoreNumber += 5;

            MyScoreText.text = "Score:" + " " + ScoreNumber;
            //The scorenumber changes
        }

        if(CoinAndEnemy.tag == "Enemy1")
        {
            ScoreNumber += 10;

            MyScoreText.text = "Score:" + " " + " " +
ScoreNumber; //The scorenumber changes
        }
        if (CoinAndEnemy.tag == "Enemy2")
        {
            ScoreNumber += 5;

            MyScoreText.text = "Score:" + " " + " " +
ScoreNumber; //The scorenumber changes
        }
    }
}

```

```
 }  
 }
```

Timer:

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

```

using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;

public class Timer : MonoBehaviour
{
    float currentTime = 0f;
    public float startingTime = 180f;

    [SerializeField] TextMeshProUGUI countdownText;
    void Start()
    {
        currentTime = startingTime;
    }
    void Update()
    {
        currentTime -= 1 * Time.deltaTime;
        countdownText.text = currentTime.ToString("0");

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

SceneManager.LoadScene(SceneManager.GetActiveScene().name);
//reseteamos el nivel una vez dañado
        }
    }
}

```

Zombee Bullet:

```

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

```

```

public class ZombeeBullet : MonoBehaviour
{
    //we need a variable to know the speed
    //Otra variable para saber cuento va aguantar ese Bullet

    public float speed = 2;
    public float lifeTime = 2;

    private void Start()
    {
        Destroy(gameObject, lifeTime); //un destroy que
        destruye el gameobject pasado el lifetime
    }

    private void Update()
    {
        transform.Translate(Vector2.down * speed * 
        Time.deltaTime); //this is because the bullet has to go in
        a down direction
    }
}

```