Proyecto

Proyecto de Desarrollo de Videojuegos

Ruben Rodriguez
Haolin Zheng
SMX-2C

Resumen del Proyecto

La idea del proyecto es recrear el conocido Super Mario Bros NES en el motor Godot y que sea modificado para dar una pincelada de creatividad y que no quede en simplemente “Mario Godot”. El protagonista de la historia no será Mario, en su lugar se controlará a Bowser y tendrá que infiltrarse en el castillo de la princesa con el objetivo de secuestrarla.

El objetivo de este proyecto es investigar, entender y desarrollar el funcionamiento interno de un videojuego, para así adquirir conocimientos básicos sobre programación y en este caso orientada a objetos, y sobre otros aspectos más artísticos como el dibujo, la música o el proceso de diseño de un videojuego.

Para lograr dicho objetivo se usará Godot y el lenguaje de Gdscript, para el aspecto artístico se tomarán fuentes de Internet y se “personalizarán” para ajustarlas al proyecto ya que el equipo no cuenta con artistas, se estimó que el tiempo para adquirir las habilidades artísticas necesarias era ampliamente mayor al tiempo disponible. Respecto a la organización del trabajo, el proyecto será subido a github y el equipo trabajará en conjunto tanto en casa usando la herramienta social Discord, y en el centro con las horas que que este otorga

El objetivo final es como ya ha sido mencionado estudiar los componentes de un videojuego y entender cómo se realiza y organiza un trabajo en conjunto en el área de la programación, también como objetivo más personal ambos integrantes del equipo desean tener una buena base para dar paso al CFGS DAM.

Palabras Clave:

Escena, Nodo, Objeto, Mario, Godot, Escena

Project Abstract

The idea of the project is to recreate the videogame Super Mario Bros Nes on the Godot Engine and modify it to give it a creative touch so it doesn't end just like “Godot Mario”. The protagonist of the story will not be Mario, instead Bowser will be controlled and he will have to infiltrate on the princess castle and rapture her.

The objective of this project is to investigate, understand and develop the inner “gears” of a videogame, in order to get the basic knowledge of programming and on this case Object-Oriented, and furthermore over other aspects more art related like drawing or music or the design process of a videogame.

To reach such objective, the Godot engine and the programming language Gdscript will be used, for the art aspect online sources will be taken and will be tweaked for the game needs given that the team has no art knowledge, it was estimated that the time to reach the required art skills were outstandingly bigger than the available time. About the project organization the project will be uploaded in Github and work the team will work together in both at the house through the social tool known as Disord. and in the center with the time that it has been otorged.

The final objective is, as it has been already mentioned, to study the main components of a videogame and understand how a workgroup is organized in the programming area. In a more personal area both members have the idea of getting a solid base for the “CFGS DAM”.

Key Words:

Platform, Node, Object, Mario, Godot, Scene

Índice:

Introducción

La idea de tomar el rol del villano como protagonista no es innovadora , pero nunca se ha visto en Mario Bros y puede traer una serie de mecánicas y perspectiva refrescante e interesante, en vez de ser el ágil pero frágil Mario, se controlara al peso pesado de Bowser que se abre paso con su inmensa fuerza. Además Bowser tiene un gran reparto de poderes y habilidades que no tiene nada que envidiar a Mario. Siempre es divertido controlar el antagonista y disfrutar su poder destructivo y porque no, hacer el mal con él.

Para esto se otorgara a Bowser una variedad de poderes que harán sentir al jugador una sensación de invencibilidad, pero si se usa de estos Bowser sufrirá consecuencias que lo pondrán en riesgo, además los obstáculos estarán hechos a su medida.

El desarrollo se hará sobre el anteriormente mencionado Godot, un motor de videojuegos open source que tiene una gran comunidad y una valoración muy alta, se utilizara el lenguaje creado por los desarrolladores de dicho motor llamado Gdscript que tiene una sintaxis inspirada/similar a la de Python y Lua. Al ser creado para dicho motor por obvias razones otorga una gran flexibilidad y optimización para dicho motor.

Para la creación necesaria del arte se utilizará algún software de pixelart, pero aún no se han estudiado las posibilidades y lo que ofrecen.

El sonido es un campo ya muy alejado del equipo y tomaría mucho tiempo adquirir el conocimiento suficiente para aplicarlo, se utilizarán sonidos ya creados por la comunidad y el material original de Nintendo. Aunque sin duda se estudiará este aspecto de los videojuegos también ya que al final es el objetivo principal.

Contexto

Uno de los mercados digitales más grandes que ha traído este siglo ha sido el de los videojuegos. Tras ver el impacto y crecimiento económico de los videojuegos se consideró estudiarlo para obtener conocimientos que podrían llegar a resultar de utilidad en el futuro cercano.

Justificación

Ambos miembros aunque interesados en el campo de la informática a nivel general están interesados en especializarse en el aspecto de la programación, así que se propusieron el reto de desarrollar un videojuego ya que es área en la que están más familiarizados, y como se ha explicado anteriormente los videojuegos son un tema en auge que va en un crecimiento exponencial. La realización de este proyecto nos dará los fundamentos para abrirnos paso al campo de la programación.

Objetivos

Para los requerimientos que se han estimado para realizar el proyecto se decidió representarlos en una Jerarquía de “Prioridad”.

Objetivos Generales

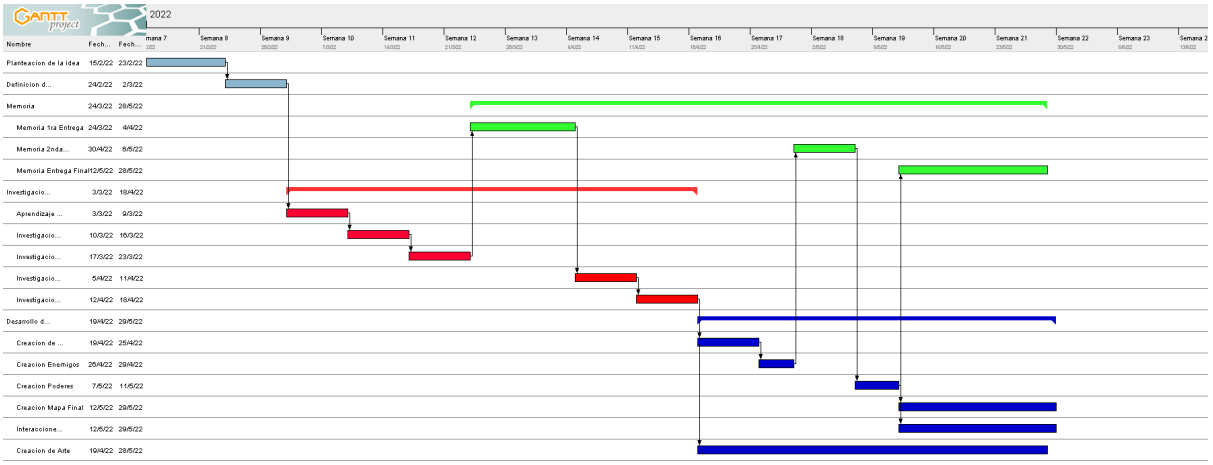
- Aprender fundamentos de programación a nivel de principiante o básico
- Aprender todas las partes básicas de un videojuego.
  (Movimiento, Salto, Aceleración, Inteligencia Artificial, Colisiones Etc.)
- Aprender funciones principales de los motores de Videojuegos
  (Sombras, Interacciones, Herramientas, Multijugador, etc.)
- Software PixelArt

Objetivos Específicos

1. Funciones básicas de Godot
2. Sintaxis de GDscript
3. Movimiento del Jugador
4. Creación del Mapa
5. Creación de obstáculos y enemigos
6. Interacciones del Jugador con el entorno
7. Animación del Jugador y enemigos

Estrategia y planificación del proyecto

La planificación se ha realizado con el software Gantt, el equipo tratará de superar los tiempos establecidos y en caso de que acabe ocurriendo aumentar el tiempo invertido diario para mantenerse en los plazos.
Más abajo si clican en la imagen podrán ver en detalle la organización del trabajo y el tiempo estimado que llevará cada tarea.



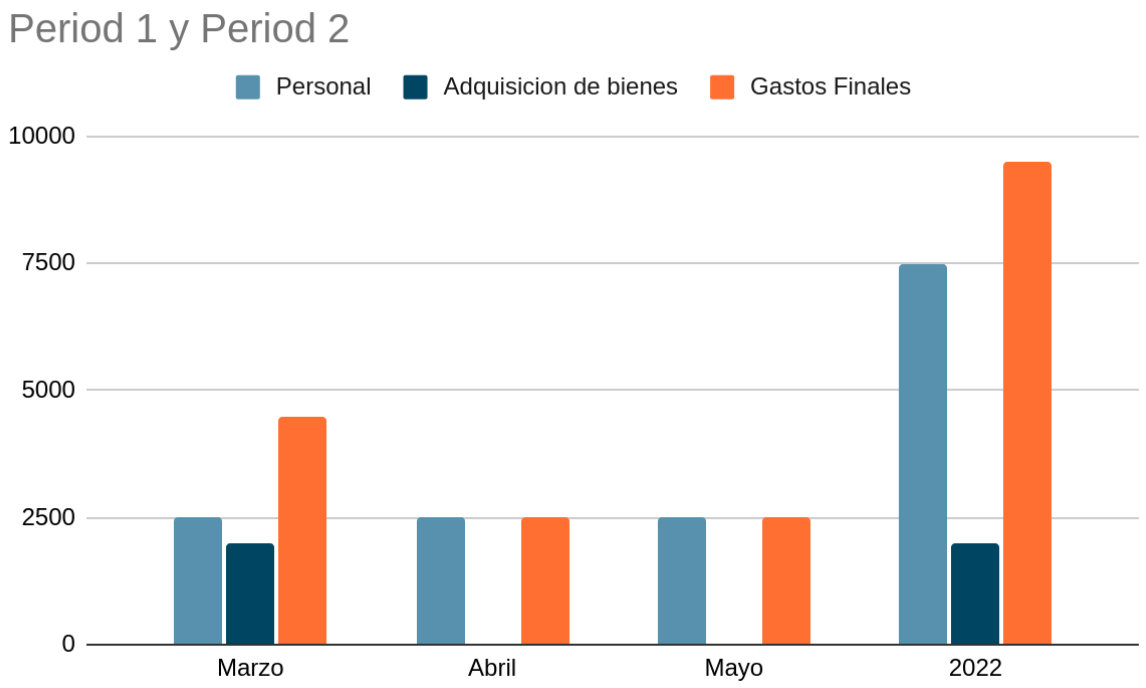
Respecto a la estrategia será un mix de crear un producto nuevo y adaptar uno existente, ya que desarrollarlo completamente desde cero no es viable ya que el conocimiento previo del equipo a la materia es básicamente nulo, así que el producto es desarrollado desde cero pero usa recursos ya creados por terceros.

Estudio económico y de presupuesto

Se ha realizado un presupuesto con el fin de calcular los gastos necessitados para la realización del proyecto

Presupuesto	**10,000€**

Gastos	2022	Marzo	Abril	Mayo
Departamento IT				
Personal	**7500€**	**2500€**	**2500€**	**2500€**
Adquisición material y bienes	**2000€**	**2000€**	**0€**	**0€**
Total				
	9500€	**4500€**	**2500€**	**2500€**



Descripción del proyecto

Análisis de Requisitos

- ➔ Fundamentos Programación y Programación Orientada Objetos
- ➔ Funcionamiento de Godot al nivel principiante
- ➔ Conocimiento basico de fisica
- ➔ Conocimiento básico de funciones lineales
- ➔ Conocimiento de diseño de videojuegos

Requisitos funcionales

- ➔ Movimiento agradable para el jugador
- ➔ Niveles con obstáculos desafiantes pero no asfixiantes
- ➔ Tener varias habilidades para tener juego dinámico y no repetitivo

Requisitos no funcionales

- ➔ Controles intuitivos y cómodos
- ➔ Diseño del nivel variado
- ➔ Jugabilidad fácil de de entender y lógica

Previsión de tareas de Investigación

Se tendría que realizar un estudio de lo siguiente:

Godot

- Crear un personaje
- Crear un Mapa
- Crear cámara
- Crear enemigos
- Crear elementos que muestran información relevante en la pantalla

    - Vida del jugador
    - Dinero del Jugador
    - Tiempo Restante
    - Puntuación
    - Vidas restantes del jugador
- Crear enemigos
- Crear obstáculos y trampas
- Crear habilidades y poderes
- Integrar Animaciones
- Integrar música

OOP (Object Oriented Programming)

- Movimiento del jugador
- Mapa que interactúa con el jugador y con los enemigos
- Enemigos que interactúan con el entorno
- Cámara que interactúa con las acciones del jugador y eventos que ocurren en el escenario
- Animaciones que se activan cuando ocurre un evento
- Música que se activa cuando ocurre un evento

Arte

- Usar herramientas como krita o gimp para realizar el arte

Tecnologías

Tecnologías Valoradas

- Herramientas para la creación de juegos
    - Godot
    - Unity
    - Unreal Engine 4
    - Pygame
- Lenguajes de programación
    - C#
    - C++
    - Gdscript
    - Python
- Arte
    - Photoshop

    - Gimp
    - Krita

Respecto al motor de juegos se escogió Godot ya que es software libre y aparentó ser muy intuitivo de usar y que tenía las funciones necesarias para satisfacer las exigencias del proyecto, además de tener un manual detallado y una comunidad aunque pequeña muy solidaria. Además el estar instalado ya en los ordenadores del centro es un gran punto a favor.

Unity era la segunda opción más probable debido a la gran cantidad de información y tutoriales, pero al tener tantas funciones hubiera sido difícil acomodarse al motor además de que el lenguaje C y sus variantes tienen una sintaxis que puede llegar a ser confusa.

Unreal Engine 4 fue otra opción a considerar pero teniendo ya Unity que es similar y da una sensación mucho abierta para principiantes no se consideró de manera muy seria, Unreal Engine 4 parece más enfocado a profesionales del campo.

Pygame fue descubierto al investigar formas de hacer juegos con el lenguaje Python pero fue descartado rápidamente porque no pareció ser muy intuitivo a simple vista y de no contener tanta información o tutoriales como las herramientas anteriormente mencionadas.

Entre los lenguajes de programación tenemos C#, C++, Python y GDscript, las variantes de C son lenguajes que ofrecen un gran rendimiento pero al parecer son un muy mal lenguaje para principiantes ya que su sintaxis puede llegar a ser confusa. Python y GDscript son lenguajes con una gran valoración entre novicios pero no encontramos nada relevante entorno a los videojuegos relacionado a python, en cuanto a GDscript tiene una sintaxis muy fácil de entender pero al ser un lenguaje de un motor específico su información es escasa.

Respecto a los programas para el arte se consideró photoshop debido a la cantidad de funciones e información que contiene pero fue descartado rápidamente por su precio, luego también se pensó en Gimp ya que tenemos algo de experiencia con este programa y estamos familiarizados con él, luego mas adelante tambien se menciono Krita pero conociendo ya Gimp no se investigó demasiado.

Tecnologías Escogidas

- Herramientas para la creación de juegos
    - Godot
- Lenguajes de programación
    - GDscript

- Arte
    - Gimp

Godot - Fue escogido debido a ser muy “beginner friendly” y tener soporte de manera sencilla en linux además de estar ya instalado en los ordenadores del centro. También tiene un lenguaje propio llamado GDscript que también sigue la filosofía de ser fácil de leer incluso para principiantes. Es muy fácil de usar gracias a su sistema de nodos que deja adherir objetos a otros objetos en una jerarquía.

GDscript - Lenguaje escogido de manera unanime debido principalmente al estar totalmente integrado con GDscript, aunque más adelante también se descubrió que su sintaxis era muy sencilla y cómoda.

Gimp - Software similar a photoshop para editar imágenes pero que también se puede usar para crear arte, se escogió debido a ser de software libre y al estar familiarizados con él ya que se realizaron prácticas, así que era la opción obvia

Estructura del proyecto

La estructura del videojuego se puede describir como una jerarquía de tres bloques, el diseño que domina en la jerarquía y la lógica y el arte que están al mismo nivel.

El diseño del videojuego es el que ordena cómo será la lógica de este, en este bloque se deciden cosas como la sensación que queremos dar al movimiento del jugador (velocidad, poder, suavidad), la forma en la que los enemigos se comportarán acorde a las acciones del jugador, los obstáculos y trampas con los que se encontrará el jugador y como se darán las pistas para sobrepasarlos.

La lógica del videojuego es en pocas palabras la sección del proyecto en la que se programa como se comportan todos los elementos de este acorde al diseño que se ha elegido

El arte es simplemente el proceso en que se crea el arte necesario para llevar a cabo los efectos visuales adicionales que tendrá el videojuego, como por ejemplo las habilidades del jugador.

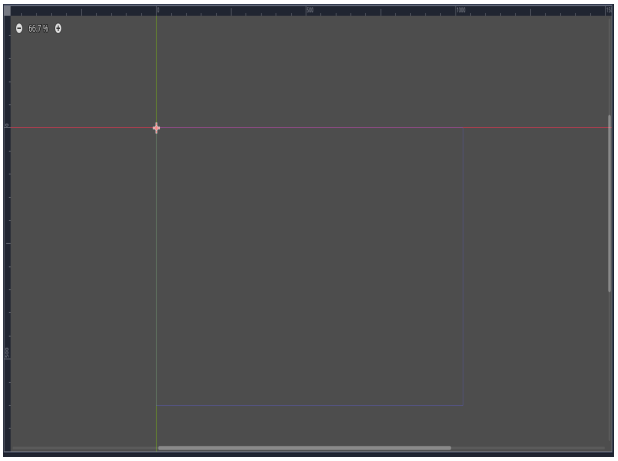
Se decidió hacerlo de esta manera ya que si el videojuego en cuestión se desarrolla sin ningún tipo de rumbo o idea el juego acabaría implementando ideas que no funcionan entre si o tendría incoherencias.

Descripción y definición de herramientas

En este bloque se describirán en detalle una estimación de los elementos necesarios para la creación del videojuego.

Escenario

Nodo 2D



Lugar donde convergen todos los elementos del videojuego para crear el escenario final

Jugador/Enemigos

KinematicBody2D

Bajo este nodo se define todos los elementos en relación al jugador o los personajes

Sprite

En este nodo se define todas las animaciones del jugador y de los enemigos

CollisionShape2D

Se define la “hitbox” de los personajes, la función de una hitbox es otorgar al jugador una forma de chocar o entrar en contacto con el resto de elementos del juego que también tengan una hitbox, es decir sin esta el personaje sería un fantasma, lo atravesaría todo.

Camera2D

Como explica su nombre es la cámara que sigue al actor, es decir son de cierta manera los ojos del jugador.

AnimationPlayer & AnimationTree

Nodos encargados de animar el personaje y dar la sensación de movimiento e interacción con los elementos. AnimationPlayer se encarga de crear las animaciones mientras que AnimationTree se encarga de especificar el proceso en el que se pasa de una animación a otra.

Habilidades

Area2D

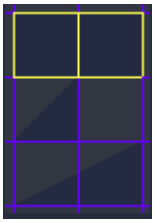
Para las habilidades se pueden usar un gran número de nodos, como por ejemplo el anteriormente mencionado **KinematicBody2D,** pero se escogerá este nodo ya que su especialidad es realizar cosas cuando entra en contacto con un objeto y no se quiere que el objeto sea movido o empujado, simplemente que reaccione.

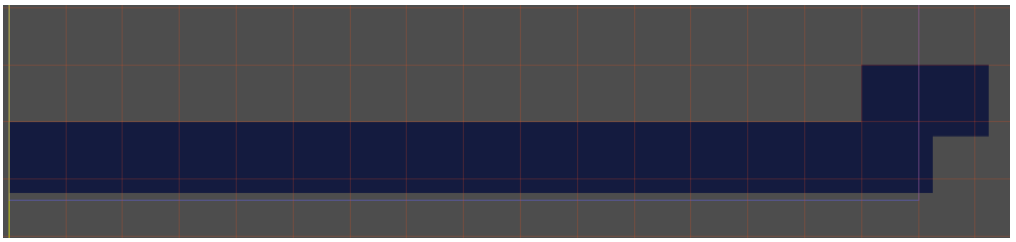
Aunque quizá a futuro se utilicen nodos como **RigidBody2D** o **KinematicBody2D** si se decide que alguna habilidad cause algún tipo de movimiento en algún elemento.

Mapa

TileMap

Un nodo poderoso que permite escoger arte de una imagen y añadirlo en forma de bloques en el escenario.



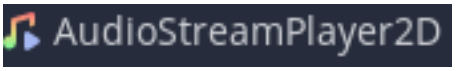


Interfaz del jugador

CanvasLayer

Nodo para la creación de interfaces o elementos más estáticos y visuales del escenario, el propósito de este nodo es mostrar por ejemplo el tiempo restante o las monedas que posee el jugador.

Música



AudioStreamPlayer2D

Nodo para la reproducción de música y sonidos

Desarrollo

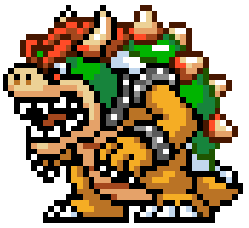
La meta de este proyecto es lograr desarrollar un nivel completo de principio a fin con cierto nivel de variedad obstáculos que a medida que se avance en el nivel estos últimos aparezcan de formas distintas para sorprender al jugador, además al mismo tiempo se le otorgará al jugador de herramientas distintas para atravesar dichos obstáculos.

Análisis de componentes del videojuego

Tras un desarrollo avanzado del proyecto se ha podido ver que el proyecto se divide principalmente en 4 grandes bloques que interactúan entre sí:

1. El Jugador o Personaje Principal

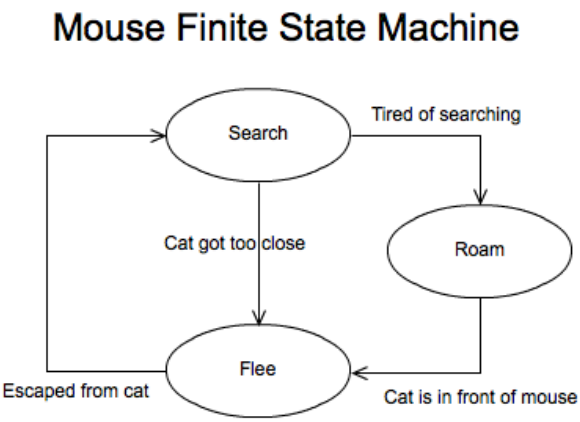
El bloque principal, que contiene mas codigo y el más importante es el propio jugador, esto es debido principalmente a que esta es la entidad que interactúa con el mayor número eventos a comparación de los demás, además la forma con la que este interactúa con cada evento puede llegar a ser radicalmente diferente, tales como recibir daño de un enemigo, caer al vacío, chocar con un bloque, colisionar con un objeto y un largo etcétera.



En este bloque se puede apreciar el desarrollo del movimiento del jugador, el salto, la gravedad, la llamada a otras entidades como las habilidades por ejemplo, reacción a ser dañado, reacción al interactuar con enemigos, etc.

2. El Enemigo o Personaje Secundario

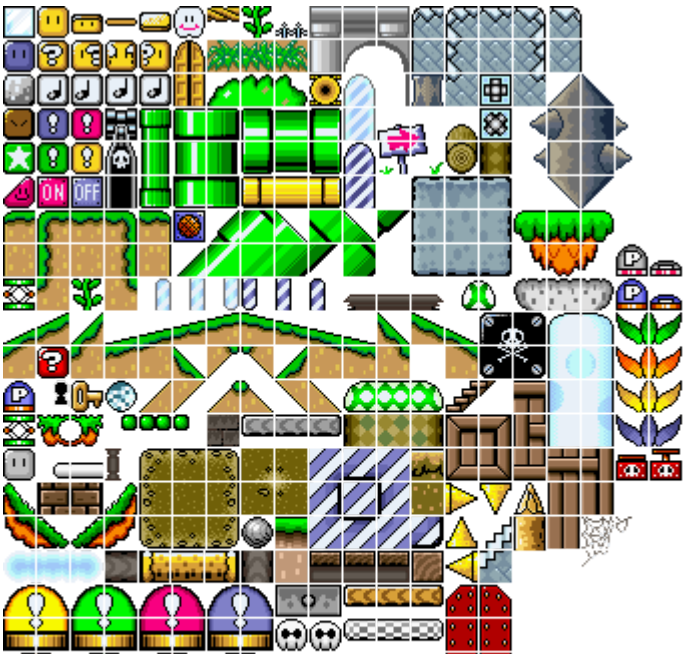
El enemigo es el segundo bloque con mayor código o incluso puede llegar a superar al propio jugador en algunos casos, esto es debido a lo siguiente:



Lo que está viendo en la imagen superior es un diagrama de una IA simple como ejemplo, es decir programar una IA es lo mismo que crear un diagrama, cuanto más compleja sea la IA de mayor tamaño será el diagrama. Se debe programar que cuando se cumplan ciertas condiciones el enemigo reaccione de cierta manera y cuando no se cumplan reaccione de otra manera. En el caso de un videojuego plataforma 2D con el estilo de Mario Bros la IA puede llegar a ser algo ligeramente más complejo que el diagrama anterior mostrado, se entrará en más énfasis en puntos posteriores.

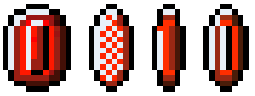
3. El Mapa

En este bloque se desarrolla el escenario en sí mismo, en este caso la cantidad de código que se implementa es casi nulo, les mostrare directamente porque de nuevo con una imagen:



Lo que está viendo en esta imagen es directamente una de las fuentes que hemos usado para desarrollar este proyecto, como puede apreciar la imagen se divide en cuadrados para dar mayor control sobre la personalización del mapa que el desarrollador desea crear, usted elige el cuadrado y ya se puede poner a **pintar** el escenario como le plazca, esto tiene un problema, el escenario más allá de las colisiones (las colisiones se encargan de que dos objetos choquen entre si) no se puede personalizar nada debido a que no es más que un cuadro enorme, al ser creado todo con la misma herramienta si añadiese código todo el escenario se vería afectado por dicho código, cosa que ningún desarrollador desea, esto afortunadamente tiene solución pero será explicada posteriormente.

4. Extras



Aquí entra cualquier entidad que no entre en los otros tres bloques pero sea fundamental para el funcionamiento del proyecto, Aunque se podría cambiar el nombre de este bloque por “Objetos” o “Ítems” y sería mayormente correcto, pero como hay **excepciones** se podría llegar a confundir entidades de este bloque con otros.

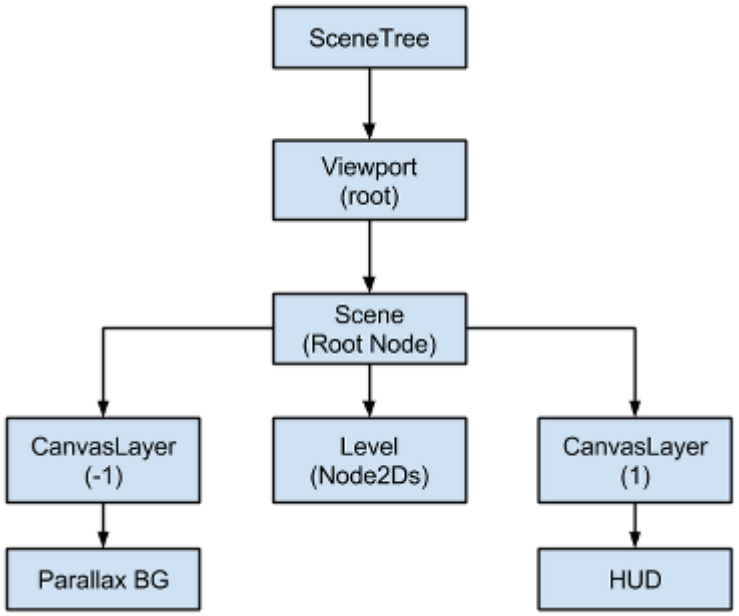
Para simplificar en esta sección entran todas las entidades que interactúan con el jugador principalmente pero no requieren tanto código como alguien perteneciente al bloque “Enemigo” y necesitamos que tengan más características que las entidades del bloque “Mapa”, en la imagen inferior puede ver una lista de ejemplos que podrían entrar en esta sección.

Explicación de Godot

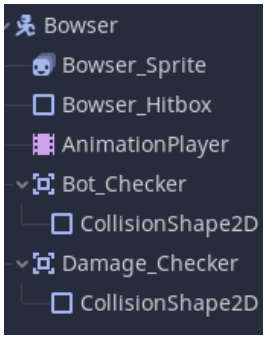
Ahora que hemos explicado los engranajes de nuestro proyecto, vamos a explicar las partes fundamentales de Godot y por lo tanto de nuestro proyecto para mayor comprensión del desarrollo de nuestro proyecto

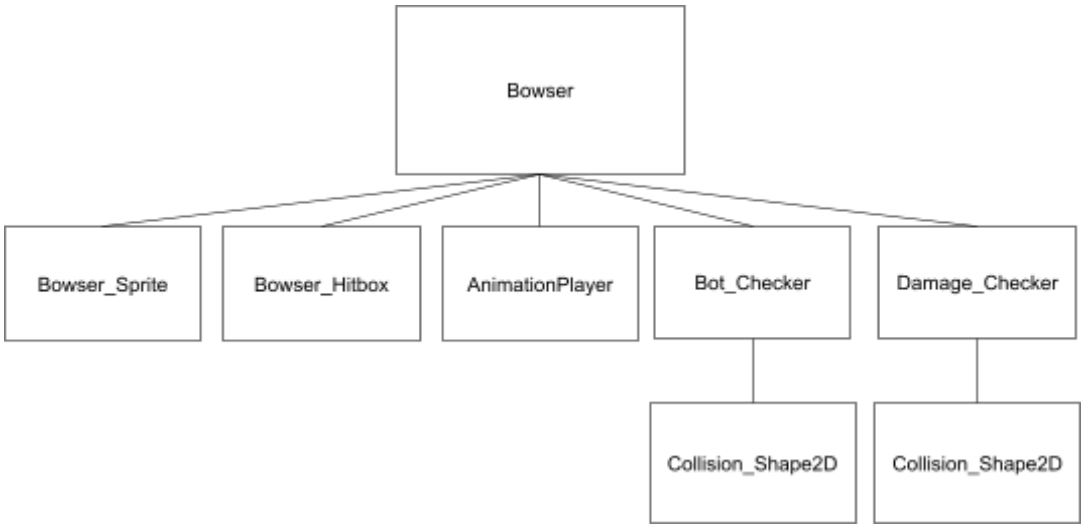
1. Nodos, Escenas y Árbol Jerárquico

Cada entidad o objeto que hemos mencionado anteriormente en Godot se puede ver como un árbol jerárquico como el siguiente:

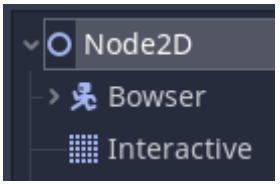


Cada rectángulo en esta imagen es un **Nodo,** un **Nodo** es una herramienta que tiene el propósito de otorgar una funcionalidad, un conjunto de **Nodos** da lugar a una **Escena**, estas **Escenas** se pueden añadir a un árbol jerárquico mayor, para mayor comprensión, les mostraremos directamente la **Escena** del Jugador de nuestro Proyecto.





En el lado izquierdo puede apreciar directamente el árbol de nuestro personaje Bowser, en el derecho para mayor simplificación y comprensión hemos creado un diagrama para que pueda ver de manera más clara la información de la izquierda, así es como se ven todos los objetos del proyecto, cada **Nodo** aporta una funcionalidad, el conjunto de estas crea la Escena. Además como hemos mencionado antes estas escenas se pueden añadir a un árbol jerárquico mayor.



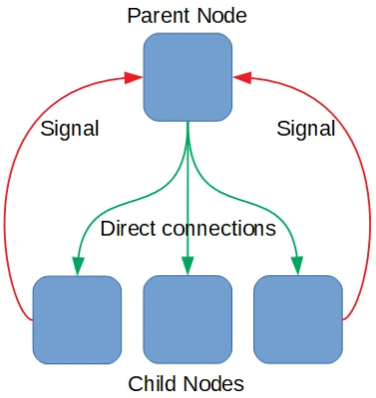
Los nodos hijos de “Bowser” no han desaparecido, Godot no los muestra para no llenar la pantalla de Nodos.

Cabe añadir que podemos **añadir** o **quitar Escenas/Nodos** con código, esto es de gran utilidad para situaciones como erradicar o hacer aparecer un **Enemigo** o **Objeto.**

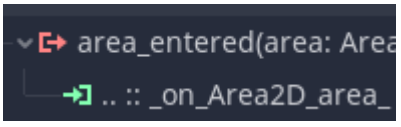
Para resumir el conjunto de **Nodos** crea la **Escena,** el conjunto de **Escenas** crea el proyecto.

2. Señales

La segunda parte fundamental de nuestro proyecto son las **Señales**, están permiten que los **Nodos** se comuniquen entre sí, en esencia una **Señal** es una notificación(aunque cabe añadir que dicha notificación se puede enviar junto con información), cuando dicha notificación llega al Destinatario este ejecutará un bloque de código o **Función**



Gracias a estas **señales** podemos hacer que por ejemplo nuestro personaje muera al recibir daño o gane poder al encontrarse con un objeto, se pueden conectar a través del editor.

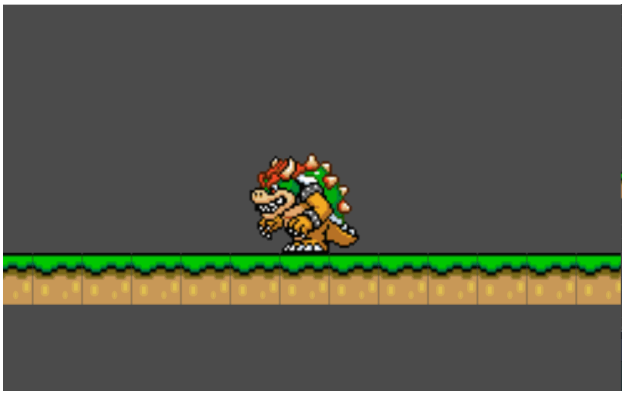


O a través de código, en la mayoría de casos es preferible hacerlo a través de código, esto se debe a que anteriormente he mencionado que podemos añadir y eliminar **Escenas/Nodos** del árbol, para usar el editor es necesario que la **Escena** esté presente desde el principio es una necesidad, con código no.

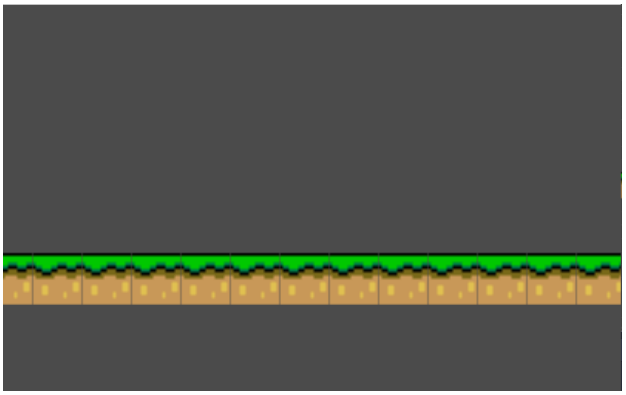
3. Colisiones

La tercera función son las Colisiones, estas permiten que dos objetos choquen entre sí, sin estas lo único que estamos programando son literalmente imágenes, para mayor entendimiento pondremos dos GIFs que muestran directamente la diferencia.

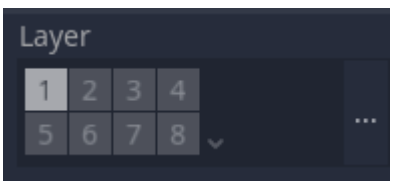
Con colisiones

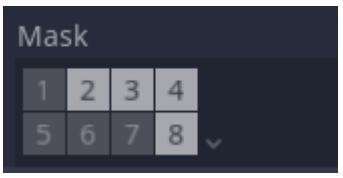


Sin colisiones



Ahora explicaremos dos conceptos de gran importancia de las colisiones. Estas són la **máscara** y la **capa**





La **capa** nos permite darle una “identificación” a la colisión de un objeto, mientras que la **máscara** nos permite decirle al objeto que identificaciones debe tener en cuenta, sin ir más lejos en el GIF superior derecho no he quitado la colisión, en realidad solo he quitado la máscara del **Mapa,** provocando que la ignore y caiga al infinito, esto es fundamental ya que no

queremos que los enemigos sean capaces de recoger objetos o por ejemplo de activar trampas, o no queremos que nuestro personaje pueda ser dañado por sus propias habilidades.

4. Funciones principales

Para esta memoria el equipo se propuso enseñar la menor cantidad de código posible ya que queremos crear una memoria que todo el mundo sea capaz de leer y entender, pero es totalmente necesario por lo menos explicar quienes son los responsables de hacer que todo funcione, en vez de explicar cada linea de codigo explicaremos brevemente las funciones que más comúnmente aparecen en nuestro código y que es lo que hacen.
Antes de nada explicaremos que es una **función**, en resumen una **función** es un bloque de código que se ejecuta cuando ciertas condiciones se cumplen, así de simple, además todas las funciones tienen un paréntesis donde puedes introducir parámetros adicionales para mayor personalización.
Ahora explicaremos las funciones principales:

```
func _physics_process(delta):
```

Para empezar debes saber que un videojuego sigue los mismos fundamentos que una animación, la única diferencia es que la animación la crea un dibujante, el videojuego la maquina, una animacion no es mas que una cantidad de dibujos uno detrás de otro que ocurre en un segundo, en la animación el estándar son 24 **fps**, es decir 24 dibujos cada segundo, en los videojuegos el estándar ahora mismo son 30/60/144… y los que sea capaz de reproducir tu maquina, aunque 30 es el mínimo aceptable. En los videojuegos cada “dibujo” es un cálculo que hace la máquina.

Para entendernos en ordenadores cada dibujo se llama “frame” o **fps**

La función que ves en la parte superior es llamada cada frame y ejecuta un código, así que en sí misma podrías decir que es un bucle, esta es la función más importante ya que sin esto ningún proyecto se movería ya que la máquina no calcularía cada frame.

El “(delta)” en la imagen superior es un cálculo que realiza Godot del tiempo que se pasa de un frame a otro, luego todo el código debajo de la función se multiplica por dicho código, esto es para que en caso de que bajen los frames por segundo los cálculos se mantengan consistentes. Ahora verá una demostración más clara:

Sin usar Delta

Usando Delta


```
func _ready():
```

Esta función es sencilla, es un bloque de código que se ejecuta cuando los **nodos** inferiores de la escena ya han dado la señal de que están listos, una vez hecho esto y la **Escena** esta lista, se ejecuta el bloque de código que haya en esta función, hay otra variante a esta función que se llama "_enter_tree():", ésta se ejecuta directamente sin esperar a los **nodos** hijos

Así que esta función sólo se ejecuta una vez cuando la **Escena** entra en el árbol.

```
func _signal():
```

Esta función realmente puede tener cualquier nombre, pero es capaz de tener cualquier nombre que desee siempre que no entre en conflicto con nombres de

otras funciones, esta función es una **Señal**, las que hemos mencionado anteriormente, cada vez que se recibe una **Señal** de otro **nodo** se ejecuta el código que se encuentra en dicha función, así es como provocamos que el Destinatario "reaccione" a la **Señal** del emisor. Sin estas funciones las **Señales** no tendrían ningún valor.

Escenas principales de nuestro Proyecto

Ahora que ya se ha explicado de manera breve cómo funciona esencialmente cualquier proyecto en Godot se explicarán de manera visual las Escenas principales del Proyecto. Para mayor entendimiento en la sección "**Descripción y definición de los componentes**" se puede ver una explicación más definida del propósito de cada **Nodo**.

También debe saber que las **Escenas** han sido modificadas para explicar de mejor manera las escenas ya que de lo contrario puede llegar a ser confuso.

1. El Jugador o Personaje Principal

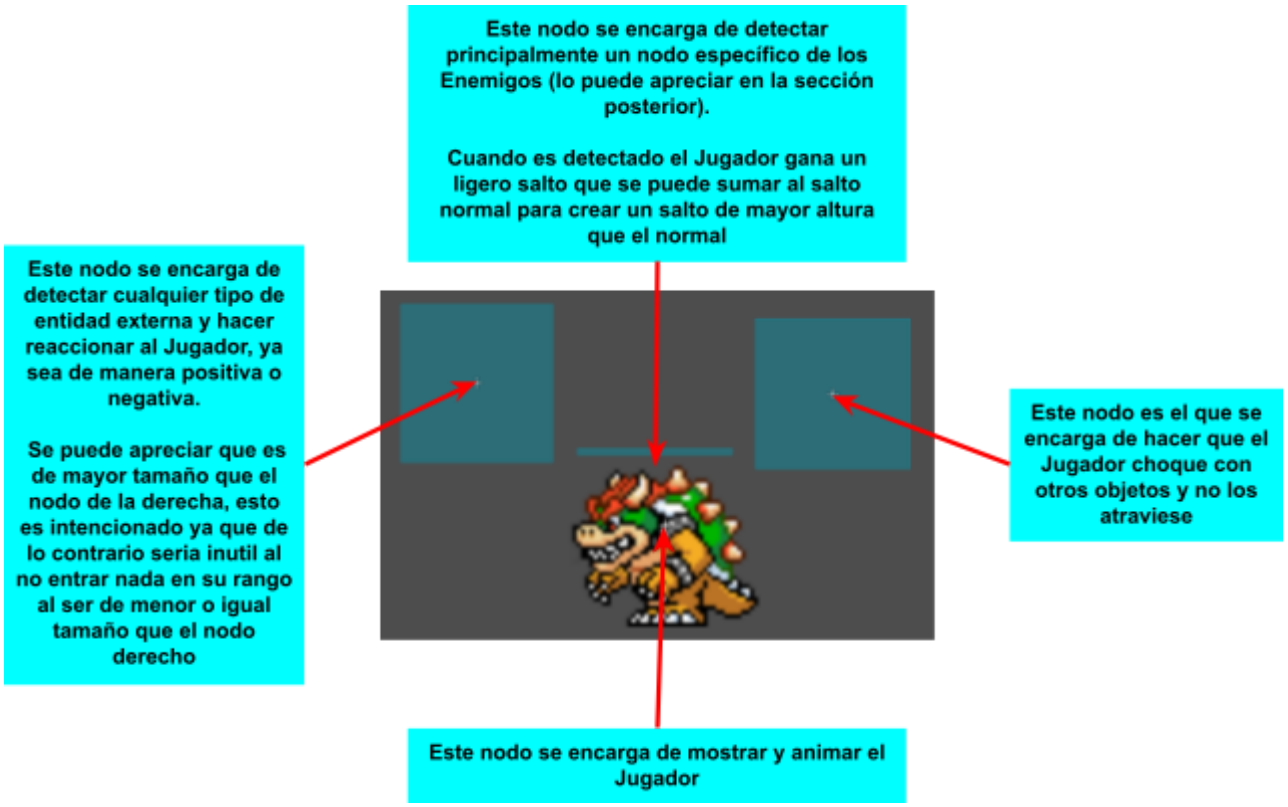
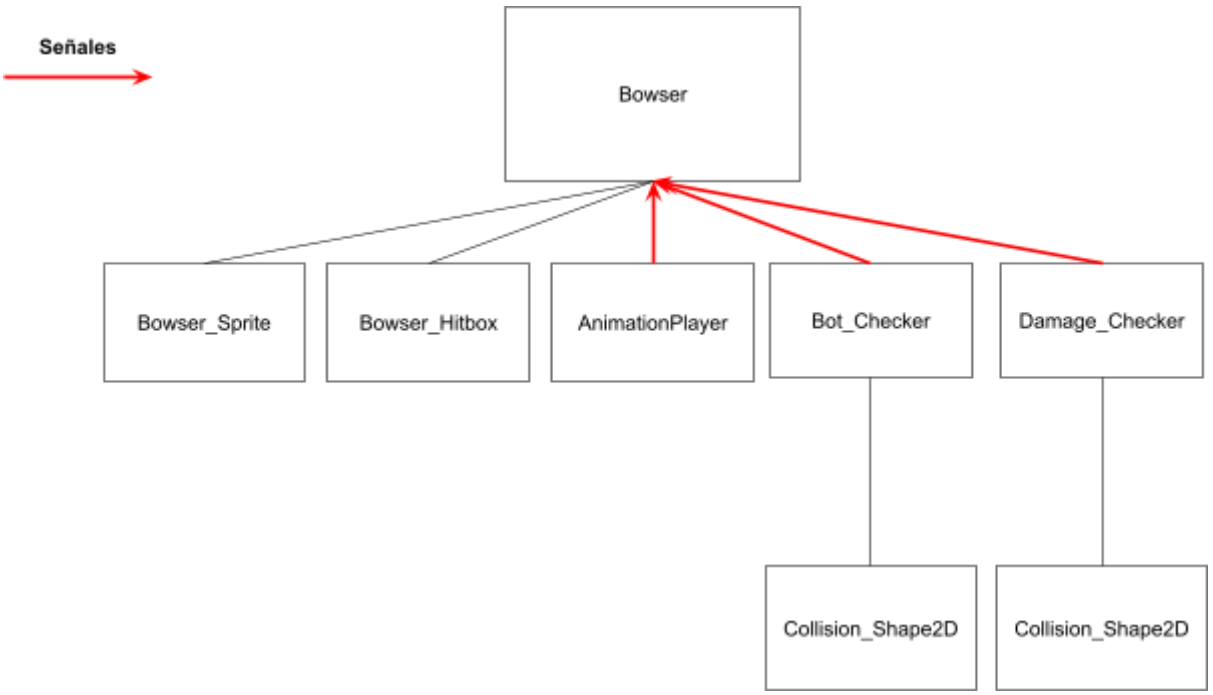


Diagrama del árbol del Jugador



Bot_Checker es el nodo señalado por el cuadro de texto izquierdo en la primera imagen, mientras que Damage_Checker es el cuadro de texto derecho, estos envían una señal cada vez que detectan algo que se les ha sido indicado

AnimationPlayer envía una señal cada vez que se acaba una animación, por ejemplo cuando la animación “morirse” se acaba se envía una señal al jugador, entonces este reinicia el nivel.

2. Escena del Enemigo o Personaje Secundario

Hay varios enemigos en el proyecto, pero todos entran en una de las dos versiones:

Version normal

Versión con mayor "inteligencia"

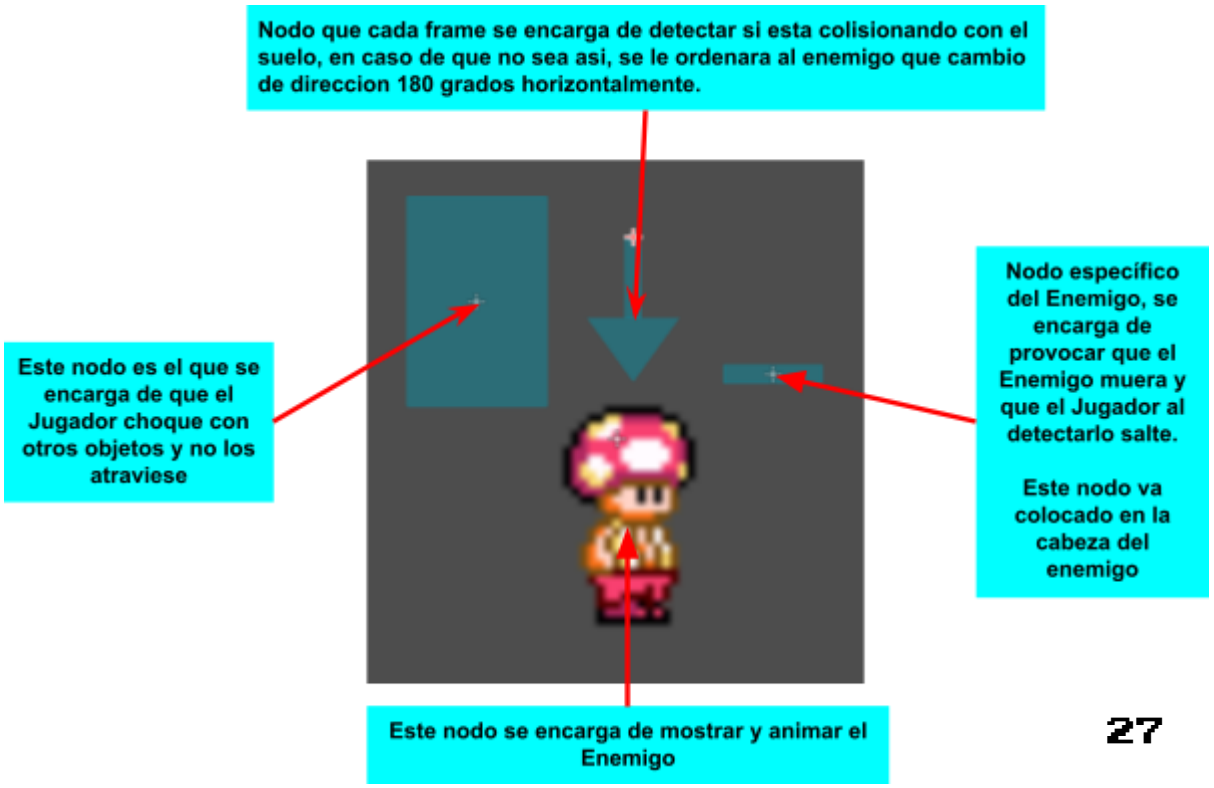
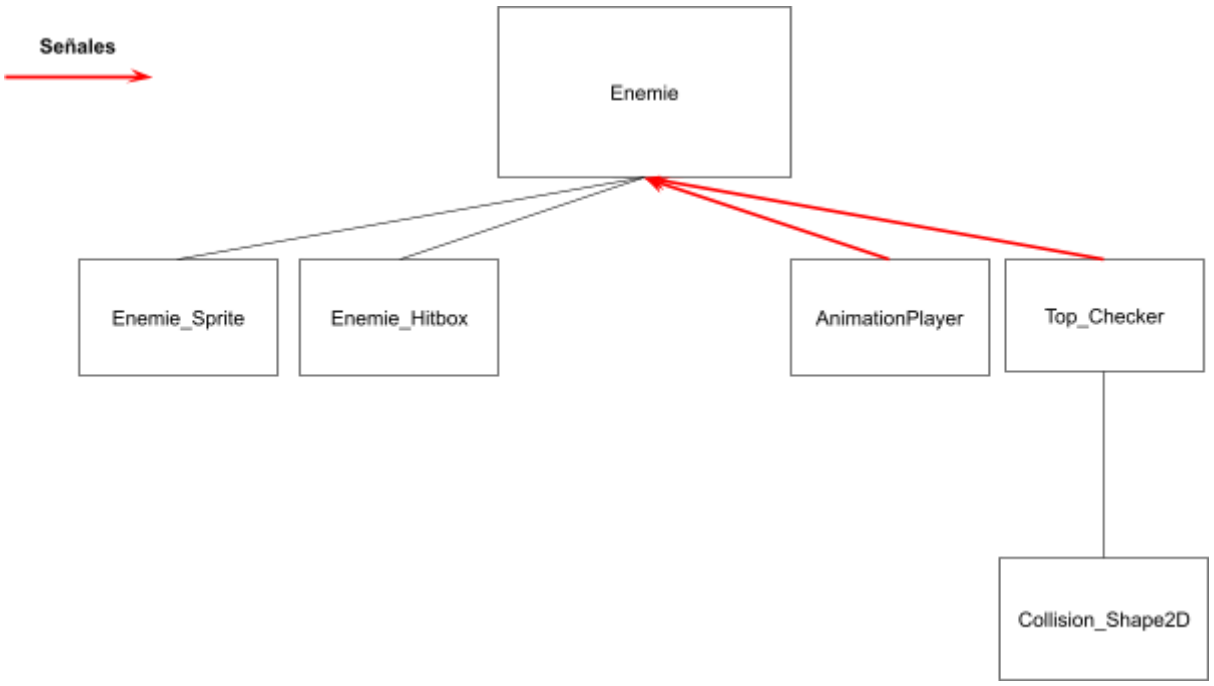


Diagrama del árbol del Enemigo

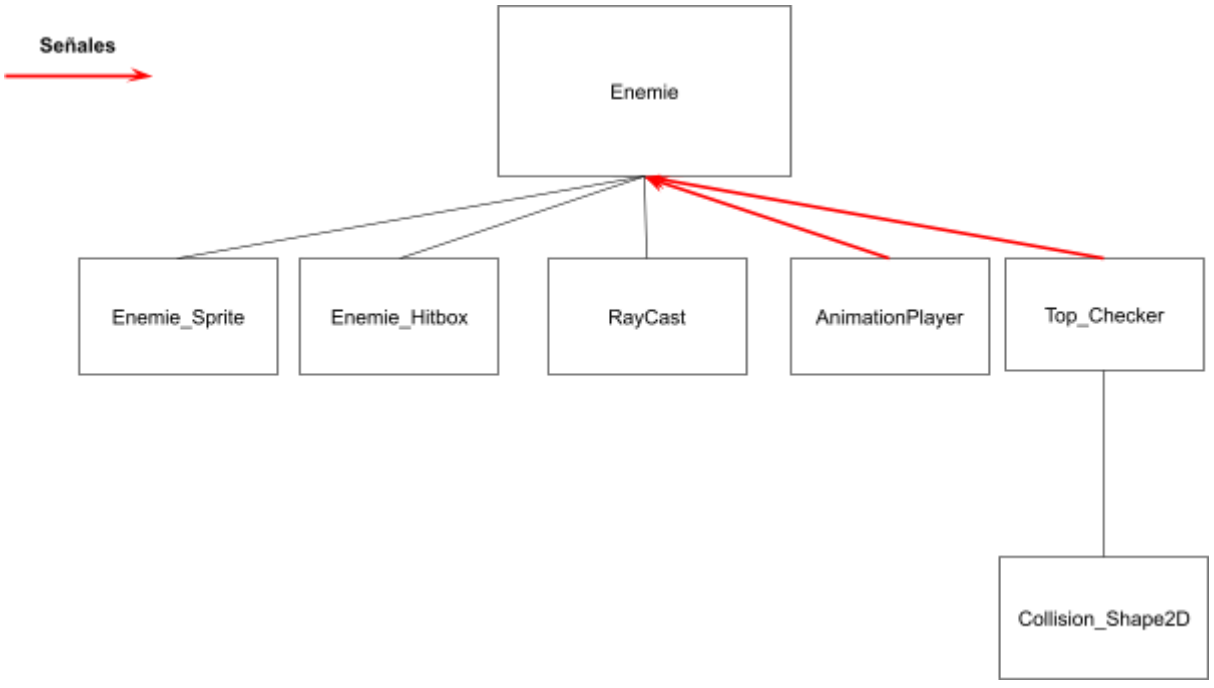
En este caso se puede dividir en **tres** versiones, son las siguientes:

Version Normal



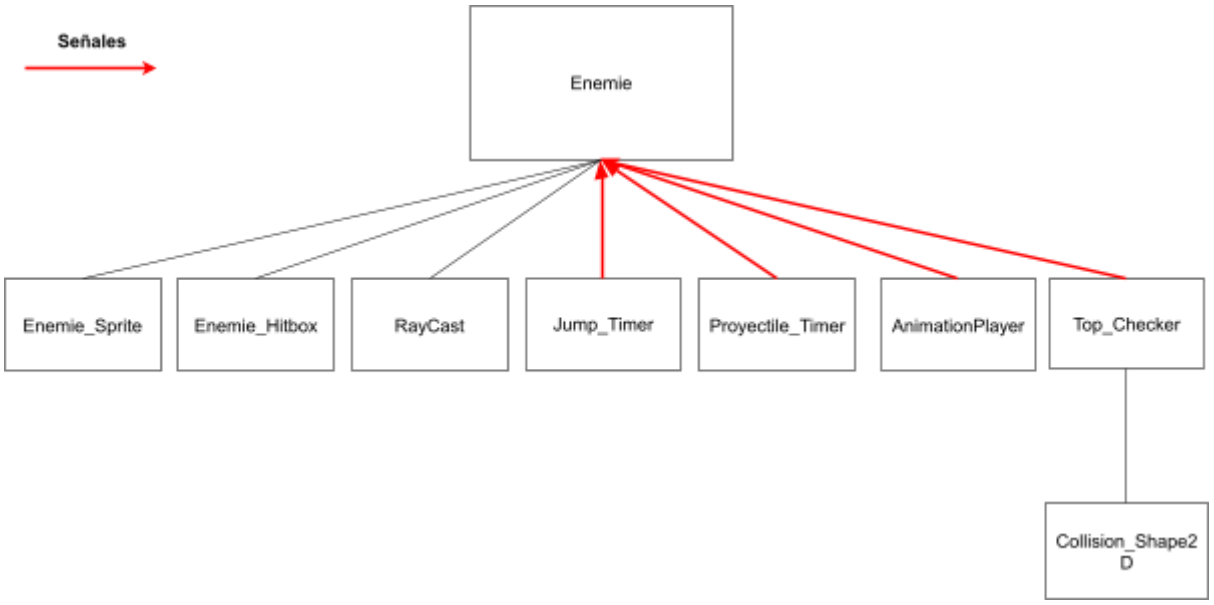
En este diagrama se puede ver de nuevo qué AnimationPlayer envía señales al **Nodo** padre**,** también se puede ver en este caso a Top_Checker, como ha sido mencionado anteriormente este cuando detecta al jugador envía una señal para ser eliminado.

Versión Inteligente



Aquí se puede ver que al Diagrama anterior se le ha sido añadido el nodo RayCast, para asi cuando no detecta suelo cambiar de dirección, quizás pensaría que este nodo enviaria una señal tambien, pero no es necesario, si se usa la función anteriormente mencionada "**_physics_process(delta):**" se puede comprobar si hay suelo cada frame

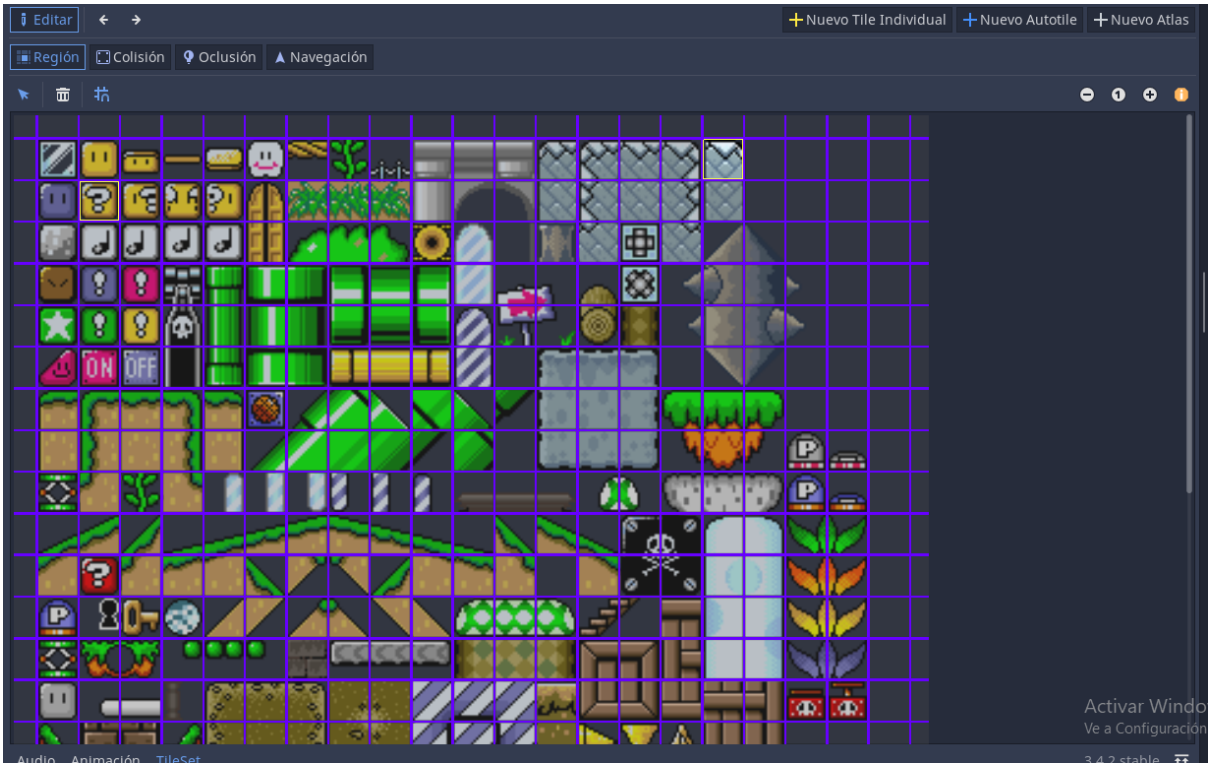
Versión Projectiles



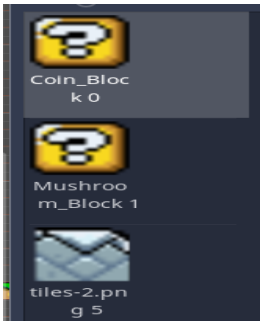
A este último diagrama se le han añadido dos nodos más, son temporizadores que cuando se acaban envían una señal al **Nodo** padre, en este caso **Jump_Timer** provoca que el enemigo salte, mientras que **Proyectile_Timer** provoca que el enemigo arroja un proyectil hacia el jugador

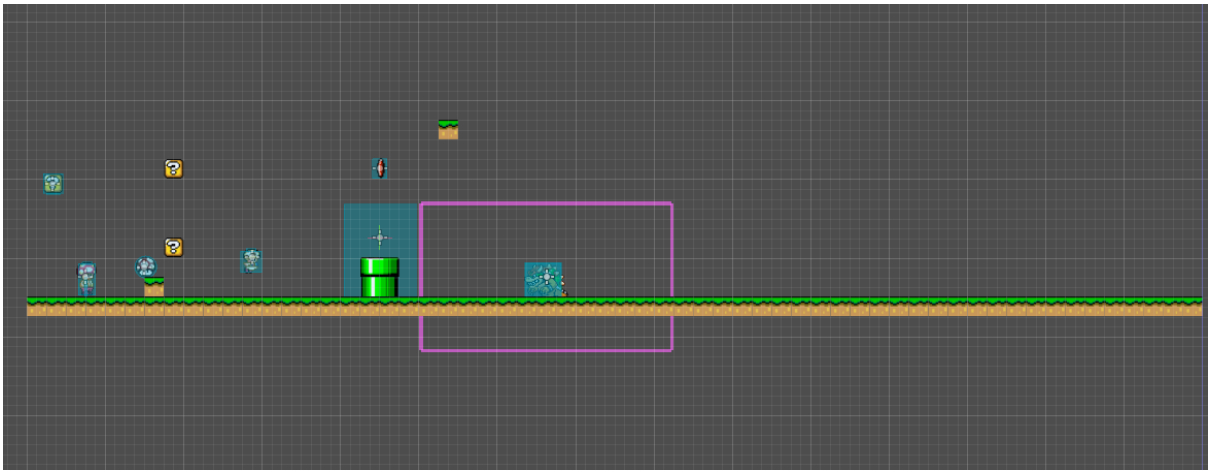
3. Escena del Mapa

El mapa debido a que no es más que una herramienta para pintar no se puede explicar de la misma forma que los dos bloques anteriores, así que se explicara cómo se "pinta" y como **ha sido mencionado en** [Análisis de componentes de videojuegos/El Mapa] no se pueden programar cada bloque distinto, aquí se explicará cómo el equipo ha solucionado este problema.



Aquí es donde se crean los bloques que desea pintar y sus respectivas colisiones, hay un gran grado de personalización pero como esta memoria no es una guía para crear un videojuego no se entrará en detalle, el propósito de esta sección es mostrar como se ve la creación de un Mapa. Tras crear los bloques necesarios usted ya puede pintar lo que le plazca, es realmente una herramienta sencilla, pero ahora se hará énfasis en cómo dar funcionalidad a esta “pintura”





Descrito de forma simple por cada bloque (que especificamos en código) que hay en la escena se obtiene la posición de cada uno, después de esto son sustituidos por objetos que entran en la sección de **Extras** anteriormente mencionada. Es decir se transforman los cuadrados del componente **Mapa** por **Extras**

4. Escena de Extras

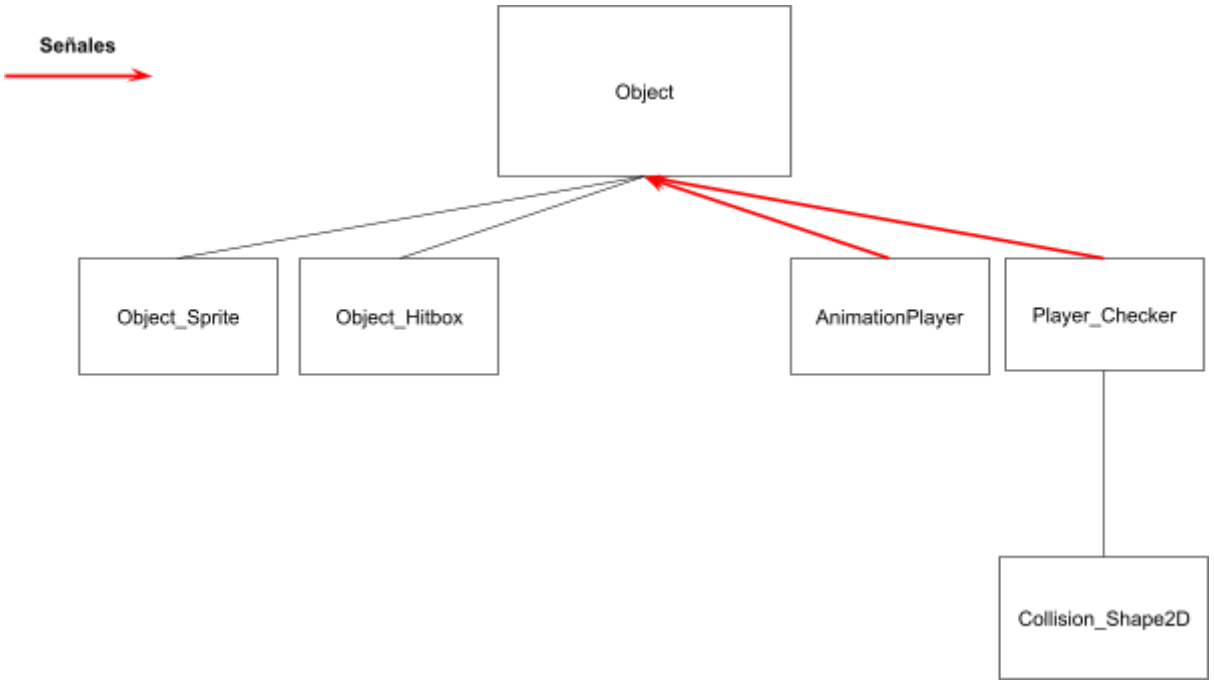
En este caso los elementos visibles en la escena son de nuevo dos tipos:

Versión con colisiones

Versión sin colisiones

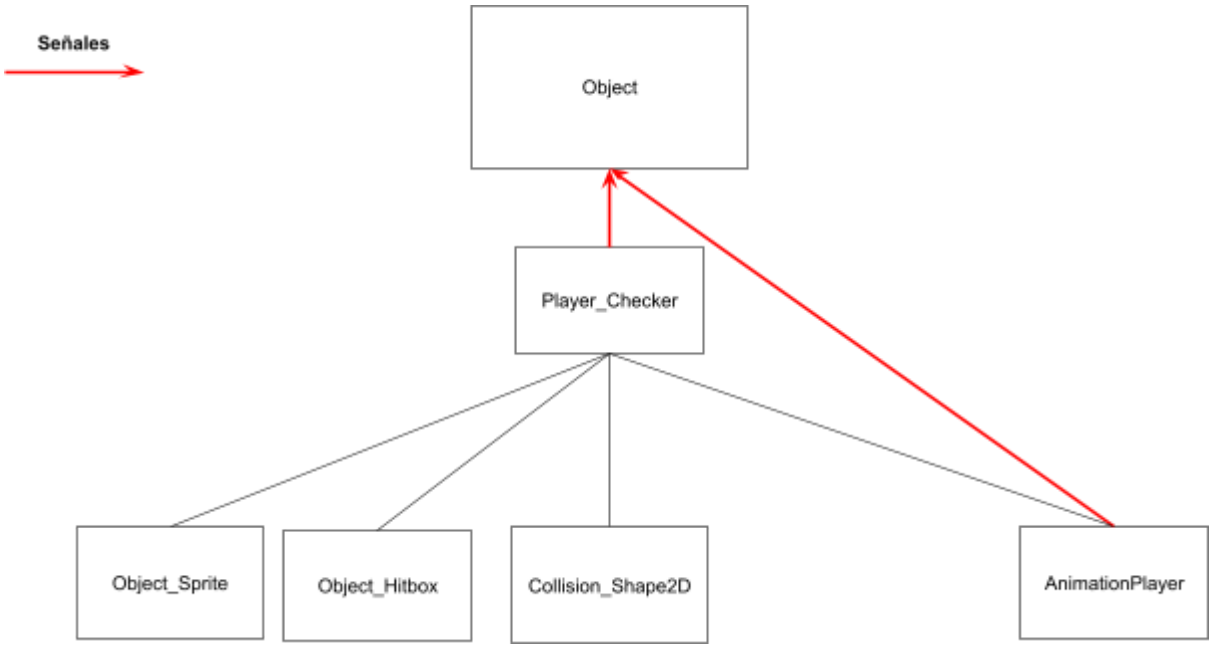
Diagrama de árbol de objeto

Versión con colisiones



De nuevo se puede ver una señal del AnimationPlayer hacia el **Nodo** padre, también se puede ver el nodo que se encarga de detectar al jugador conectando una señal con dicho **Nodo.**

Versión sin colisiones



En este caso se puede apreciar algo distinto a lo usual, en esencia es un Diagrama similar a los anteriores pero los **Nodos** hijos a excepción de uno son en realidad nietos, esto se debe a que quiere que esta escena sea un “**Area2D**” pero se requiere la movilidad de un “**KinematicBody2D**” así que para ello se crea este árbol jerárquico. (La explicación a fondo de los Nodos esta en”**Descripcion y Definicion de los componentes**”)

Trabajo a futuro

En primera línea y sin la presión de una fecha de entrega definida el equipo se tomaría el tiempo de pulir el código e investigar formas de optimizar este.

Tras esto el equipo estudiaría a fondo el desarrollo de una IA para poder desarrollar enemigos más complejos que den lugar a situaciones más interesantes. El campo de la Inteligencia Artificial era demasiado grande para estudiarlo apropiadamente en el lapso de tiempo que se otorgó a los equipos para desarrollar su proyecto.

Una vez ya se tenga una mayor comprensión de cómo programar una IA se expandiría las mecánicas del jugador para dar un mayor número de opciones a este sobre cómo proseguir en el nivel, además de que se incluirían una mayor variedad de obstáculos y enemigos

Sería necesario estudiar y desarrollar una interfaz donde se muestre la puntuación, monedas y vidas del jugador, ya que faltó tiempo para desarrollar este campo.

El sonido es otro de los componentes que el equipo no pudo desarrollar asi que seria ideal implementarlo

Tras todo esto se tendría un prototipo ideal que mostraría las habilidades informáticas del personal y podría servir como prueba de nuestras habilidades para promocionarse hacia empresas que requieran programadores.

Tras la finalización definitiva del proyecto el equipo se encargaría principalmente de seguir puliendo sus habilidades como programadores individualmente en otros campos para no quedarse encerrados en uno y enfocarse en el estudio en el Ciclo Superior

Conclusiones

La conclusión final es que ha sido un proyecto arduo pero muy interesante, ha sido un gran desafío debido a que los mencionados integrantes tenían un conocimiento previo al proyecto de **Cero,** esto ha provocado que constantemente las expectativas del resultado final han tenido que ir bajando a medida que el tiempo avanzaba, aún así el equipo no tiene ninguna duda de que las horas invertidas en el proyecto son más que suficientes para llamarlo un credito de sintesis en toda regla.

Sin lugar a dudas la parte más ardua de la realización de este proyecto a sido de hecho el núcleo entorno al cual gira toda la motivación del proyecto, ya que el lenguaje escogido aunque similar a otros la cantidad de recursos de aprendizaje de estos es escasa, y cuando se trata de indagar en componentes más complejos del proyecto ya se llega a una cantidad de información cuasi nula.

También otra de las frustraciones del equipo es la realización de que el código en si esta lejos de ser profesional o una calidad satisfactoria aunque esto es razonable debido a lo explicado ya previamente y al hecho de que es un objetivo imposible de lograr con tres meses de preparación además de que uno de los integrantes tenía que hacer las prácticas dificultando aún más el proyecto. A pesar de esto el equipo creé que esta experiencia será una base sólida para el ciclo superior de “Desarrollo de Aplicaciones Multiplataforma” que es el siguiente “nivel” al que el equipo se va a enfrentar, lo cual era uno de los objetivos principales de los integrantes con el desarrollo de este proyecto.

Tristemente el proyecto no ha llegado al nivel de maduración esperado debido a todos los inconvenientes durante la realización de este, el más grande sin lugar a dudas era la FCT tomando una parte gruesa de las horas disponibles del personal. Pese a esto el proyecto ha llegado a un nivel que el equipo califica como satisfactorio como entrega final ya que se ha realizado un juicio objetivo de proyectos terceros y tienen confianza de que el calibre del proyecto es comparable al de estos que han tenido éxito aunque aún así lamentan no haberlo podido desarrollarlo hasta el final.

Otro gran limitante sin ningún tipo de incertidumbre era que el equipo no tenía entre sus filas el conocimiento o talento artístico, dificultando y limitando gravemente las posibilidades y ideas que se tenían con este proyecto, aun así el equipo trabajó con lo que tenía entre las manos y hizo lo mejor que pudo con los materiales existentes.

A pesar de todo esto el equipo está satisfecho con el resultado final y ha sido una experiencia distinta a lo usual, divertida a veces, frustrante comúnmente pero definitivamente interesante en todo momento y que ha dejado una sensación de

querer expandir más las fronteras y el conocimiento del equipo y se tiene la esperanza de que la calidad del trabajo solo siga incrementando indefinidamente.

Problemas y Errores

Problemas

Incomunicación

- Este ha sido uno de los mayores problemas que ha sufrido el equipo, la raíz de este problema proviene del hecho de que tenían horarios totalmente distintos principalmente debido a que uno de los miembros acababa de empezar las FCT (Formación en Centros de Trabajo) y el otro miembro las había finalizado recientemente, provocando un desbalance, aun así ambos miembros hicieron todo lo posible para la realización del proyecto.

Cero conocimiento previo y escasos recursos de aprendizaje

- Este sin lugar a dudas el talón de Aquiles del proyecto, la solución más óptima que el equipo encontró fue usar recursos de origen inglés para obstáculos de nivel “principiante” y cuando se buscaba soluciones a problemas complejos se acudía a la extensa documentación de Godot que viene ya integrada de por si en el editor.

Errores

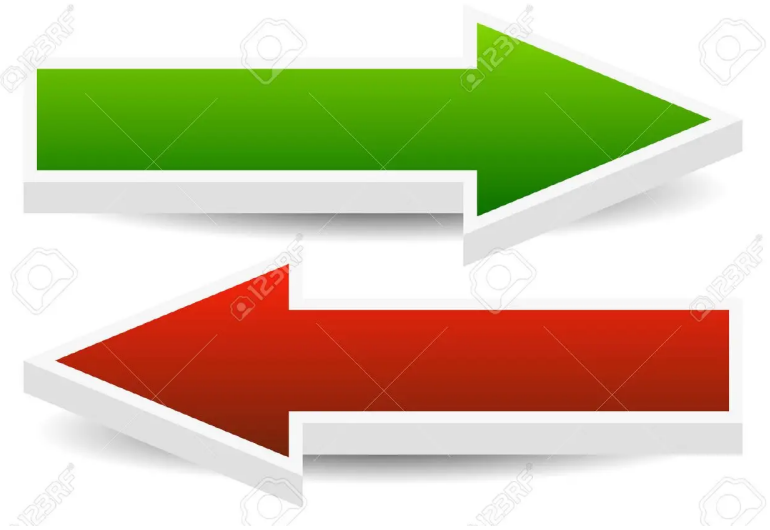
Durante el desarrollo hubo una gran cantidad de errores menores, mencionaremos los más importantes:

Dificultades en el desarrollo del movimiento del jugador

Esta sección del desarrollo fue la más costosa en términos de tiempo debido a que el equipo aún se estaba acostumbrando tanto a programar como al lenguaje de programación GDScript además del propio editor y su sistema de nodos, además al mismo tiempo se tenía que pensar cómo se iba a desarrollar el proyecto y la memoria. En un videojuego plataforma 2D el movimiento del jugador es uno de los núcleos en los que gira todo el proyecto, el más ligero cambio podría provocar un reajuste del proyecto entero así que la cautela era una necesidad en esta sección. A medida que fue avanzando el primer mes se logró tener mayor confianza con el código y desarrollar el proyecto más rápido, y se logró obtener un jugador que cumpliera las exigencias del equipo. Al principio del proyecto se usó un

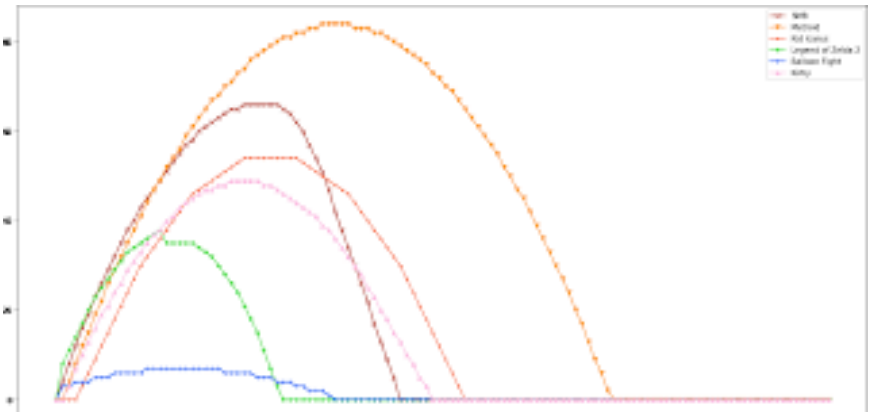
cubo para ir poco a poco logrando aprender a codificar las distintas mecánicas de un jugador como son:

Movimiento hacia los lados



Salto y Gravedad

Control de Salto



Y más mecánicas que no se van a mencionar debido a que ya no se encuentra en la sección de desarrollo.

Dificultades para programar obstáculos o el mapa

Aquí puede ver en detalle cual es el problema
Aquí puede ver en detalle cómo se llegó a su solución

Error que no guarda el estado del jugador al cambiar el nivel

Al cambiar de escenario todos los cambios ocurridos durante el nivel son borrados ya que no están guardados en ningún lado. Esto es solucionado definiendo un documento donde se guarden los cambios ocurridos durante el nivel, al cambiar de nivel el proyecto leerá dicho documento y hará los cambios necesarios para empezar el nivel con los cambios ocurridos en el anterior nivel.

Ha habido un número mucho mayor de errores que los mostrados además de que estos han sido simplificados y agrupados para simplificar la explicación, pero si se describiera cada uno y en detalle fácilmente la memoria se vería duplicada en tamaño así que no es conveniente ni para el escritor ni para el lector, pero el equipo quería mostrar por lo menos que no han quedado impunes de obstáculos en el camino y que las horas invertidas son más de las que se pueden apreciar a simple vista.

Webgrafía

Godot/Programación

Diseño del movimiento del Jugador
Porque usar Godot
Godot explicado en 5 minutos
Documentación Godot
Godot Forums
Godot Basics Tutorial (No estudiado entero)
GDscript para principiantes Parte 1
GDscript para principiantes Parte 2
Introducción a gdscript (Lista de 36 videos)
Más recursos de GDscript
Tu primer juego con Godot
Desarrollo de Videojuego Plataforma Godot Parte 1
Desarrollo de Videojuego Plataforma Godot Parte 2
Desarrollo de Videojuego Plataforma Godot Parte 3
Movimiento del Jugador Godot
Más recursos sobre el movimiento del jugador
Godot Señales
Salto
Salto Godot
Bola de fuego
Mapa Interactivo
Plataformas de una dirección
Fondos
Variables Globales
Plataformas Movibles
Respawn de Enemigos
Enemigo que persigue al jugador
Programar Enemigos
Nodo AnimationPlayer
Nodo Tilemap

Recursos Artísticos

Jugador/Mapa/Enemigos